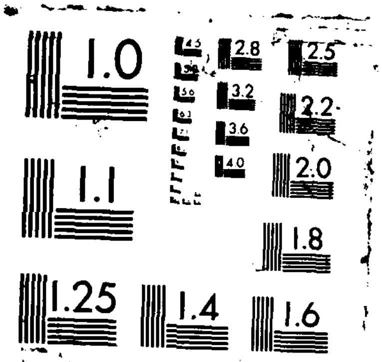
AD-A190 679 FEATURE EXTRACTION USING THE HOUGH TRANSFORM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING M L HILL DEC 87 AFIT/GE/ENG/87D-24

1/2

UNCLASSIFIED F/G 12/6 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

AD-A190 679

FEATURE EXTRACTION USING

THE HOUGH TRANSFORM

THESIS

Marvin L. Hill
Captain, USAF

AFIT/GE/ENG/87D-24

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

88 3 24 077

1

AFIT/GE/ENG/87D-24

FEATURE EXTRACTION USING THE HOUGH TRANSFORM

THESIS

Marvin L. Hill
Captain, USAF

AFIT/GE/ENG/87D-24

Approved for public release; distribution unlimited

AFIT/GE/ENG/87D-24

FEATURE EXTRACTION USING THE HOUGH TRANSFORM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements of the Degree of
Master of Science in Electrical Engineering

Marvin L. Hill, B.S.
Captain, USAF

December 1987

Table of Contents

Preface

This research began on two fronts. One was a desire to build upon the work performed by Lt Carl Tong in the area of multisensor data fusion. Lt Tong left a variety of software tools which proved useful primarily in displaying images on the Evans and Sutherland PS340 Graphics Workstation. The second was an interest in creating and implementing computer-generated interferograms. The topic of Hough transforms was originally chosen as an application for computer-generated interferograms. When difficulties in using the interferograms on complex fields arose, discrete implementations and properties of the Hough transform became the primary focus.

This project created some useful software tools for implementing and performing digital filtering on Hough transforms for use on the VAX computer. It also implemented an "in house" technique for producing computer-generated interferograms which proved invaluable to several other research projects.

There are a few individuals whom I would like to acknowledge for their assistance throughout my thesis effort. First and foremost is my thesis advisor Capt Steven K. Rogers whose continual patience, enthusiasm and prodding gave me the necessary direction and focus for my research. I would also like to thank my committee members Dr. Mathew Kabrisky and LTC James Mills for providing intellectual stimulation. Finally, I would like to thank fellow AFIT student Lt Mike Mayo for his photographic assistance.

-- Marvin L. Hill

List of Figures

List of Tables

AFIT/GE/ENG/87D-24

Abstract

This thesis applied the normal straight line parameterization of the Hough transform to a variety of images using the accumulator method. Simple inputs were used initially to illustrate the distortion characteristics of the Hough transform due to rotations, scales and translations of an input. Making use of work performed by D. Casasent and R. Krishnapuram of Carnegie-Mellon University, the Hough transform was then applied to segmented and edged doppler images. A distort-and-compare routine, which makes use of the Hough space distortion characteristics, was implemented in the Hough transform domain to estimate input space characteristics of an object.

Next, the Hough transform, generated using Fourier transform techniques, was applied to some of the same inputs to demonstrate that the accumulator method is actually a discrete version of the continuous Hough transform. An unsuccessful attempt at implementing the continuous Hough transform using computer-generated interferograms was outlined. A method of implementing the continuous Hough transform and its inverse using phase filters was presented as a suggestion for further research.

I. Introduction

A. *Background.* Mission work load in all combat areas has increased dramatically in recent years. A significant portion of combat requirements involves acquisition and identification of hostile targets within the threat environment. Thus, a considerable amount of research has been devoted to segmenting targets from a variety of backgrounds and current algorithms are often quite effective in performing this task. The ability to classify targets either before or during segmentation is a much more difficult task. Investigation, analysis and implementation of alternative feature extraction techniques is essential to ensure out-numbered forces maintain a technological edge. The Hough transform has shown some promise in its feature extraction properties. This thesis will investigate the properties of the Hough transform and research methods for its implementation.

B. *Approach and Scope.* The first task in this thesis project is to gain good working knowledge of the ADA programming language and programming techniques in general. A variety of display drivers and input/output routines were created in the Tong [20] thesis project with most of the programming accomplished in the ADA language. Thus, creating the necessary requirement of understanding current file formatting in order to obtain, create or process image files. The second task is to research current knowledge of Hough transform techniques which might suggest efficient means of implementation. Third, working edge images must be created to test Hough transform implementations and determining Hough transform characteristics. Elementary test patterns are required as well as examples of actual segmented data. In addition, computer implementations of the Hough transform techniques must be created and display

routines modified to present the output. The final task will include an investigation into possible real-time implementations of the Hough transform.

C. *Organization.* This thesis begins with a summary of current theory in Section II. A basic definition of the Hough transform is presented followed by some of the methods used for its implementation. Current attempts to obtain the transform optically are also discussed. The creation of working edge images is briefly outlined in Section III followed by a description of the two Hough transform processes in Section IV. The effects of the transform on basic shapes is analyzed in this section as well. A process for estimating input space distortions within the Hough domain is described in Section V and examples of its application to real edge images are depicted. This is followed by a general discussion of the results of the thesis in Section VI. In Section VII, a suggested continuous optical implementation technique is presented as a recommendation for further research with background data for this approach outlined in Appendix B and Appendix C. Finally, concluding statements on the research project reside in Section VIII.

II. Summary of Current Knowledge

The Hough transform was developed in 1962 by P.V.C. Hough [1] to trace the paths of atomic particles in cloud chambers. The initial procedures outlined by Hough [1] were written in terms of electrical circuit diagrams. It wasn't until 1972 (Duda–Hart [2]) that Hough transform techniques were applied to pattern recognition.

A. *Basic Approach.* Current literature applies a rather broad definition to the term Hough transform. In general, a Hough transform maps points in an input feature space (e.g., a Cartesian coordinate space) to curve shape parameters in an output space. This mapping is usually based on the coordinates of the point and possibly the directional gradient at that point.

Hough transform techniques are used to detect specific types of curves within the input space. Thus, the curve type is normally included when describing a specific type of Hough transform. Variations most often discussed include Hough transform mappings to straight line, circle and ellipsoid parameter spaces. These variations, along with generalized parameter space applications, are discussed in the following paragraphs of this section.

Implementation of Hough transforms for detecting curves within an input scene requires segmentation and edging of that input scene. This is not an overly restrictive condition since many segmentation algorithms have been developed (e.g., Tong [20] and Ruck [23]). The Hough transform can then be applied to extract features from the outline of the segmented objects as an aid to

classification.

B. *Simple Curves.* The applications of the Hough transform presented by Duda–Hart [2], [3] and Shapiro [4] describe mappings to a straight line parameter space. From the basic slope intercept equation for a line in Cartesian coordinates,

$$y = a\,x + b \tag{2.1}$$

a binary input space f(x,y) is mapped to a parameter space H(a,b). Thus all points in the input space situated along the line $y = a_0x + b_0$ are mapped to a single point $H(a_0,b_0)$ in the Hough space.

Similarly, a Hough transform parameter space developed for detection of paraboloids of the form

$$y = c\,x^2 + d\,x + e \tag{2.2}$$

would produce a Hough output space H(c,d,e). The methods used to implement this type of Hough transform vary considerably. Most, however, use a variation of the procedure used by Shapiro [5:129] which require taking derivatives at each edge point (pixel) in the input space.

C. *Hough Transform Using a R-Table.* The first useful generalized Hough transform technique was introduced by Ballard [6] in 1981. In addition to mappings to parameter spaces of analytic curves, this procedure can create a

Hough parameter space for an arbitrary shape. The transform is applied by taking directional derivatives at each edge point. This value is compared to a table describing which points in the output space to increment. Each point in the output Hough space is then incremented as shown in Figure 2.1,

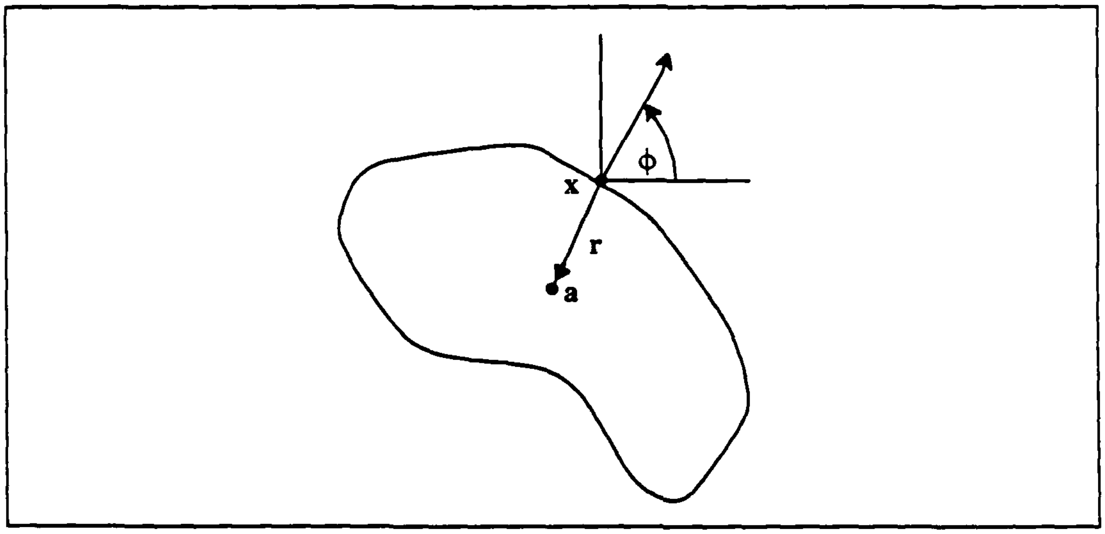


Figure 2.1. Geometry for Generalized Hough Transform[6:116]

A directional gradient is taken at each edge point **x** in the input space. The gradient value ϕ corresponds to an increment vector **r** in the R-Table. The increment vector identifies the point, **a**, to increment in the Hough space.

with the edge point in the input space, directional gradient, increment vector and increment point in the output space are represented by **x**, ϕ, **r** and a respectively. The table, referred to as an R-Table, contains a set of vectors indexed to each directional derivative of the curve to be detected. Rotations and scales of the curve in the input plane can be handled by rotating and scaling the increment vectors within the R-Table.

D. *Normal Straight Line Parameterization.* The Hough space parameterization based on the normal representation of a straight line,

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{2.3}$$

was used initially to alleviate the problems associated with infinite slopes in the alternative slope-intercept parameterization of Eqn (2.1). The geometry of this parameterization is shown in Figure 2.2.

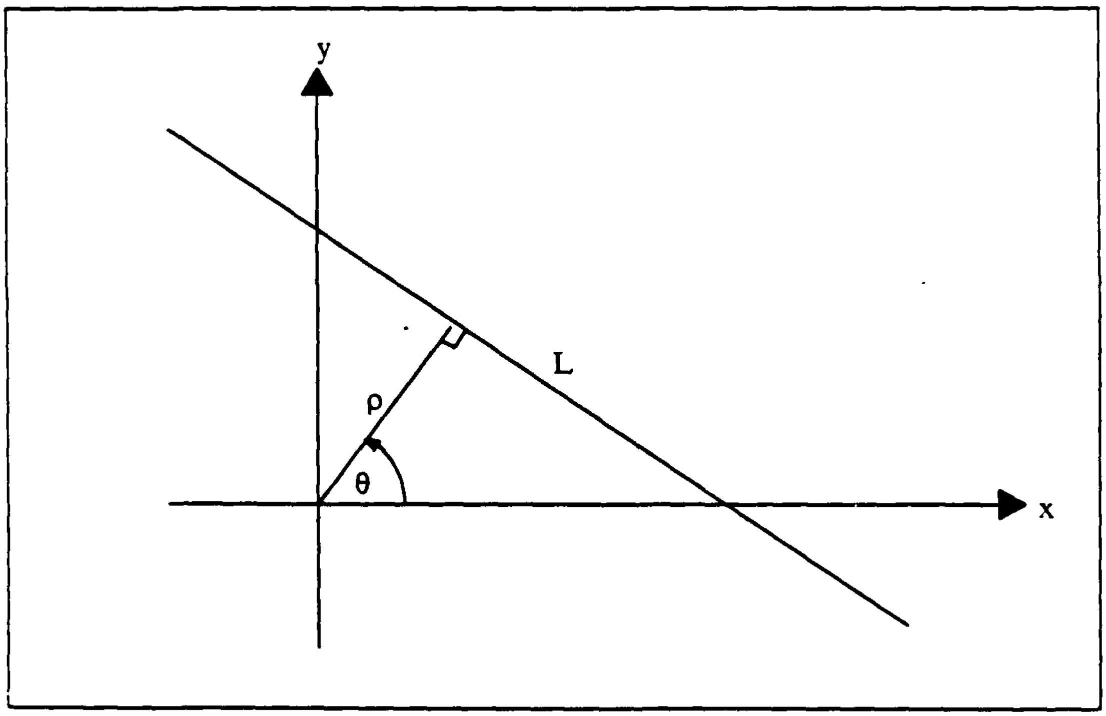


Figure 2.2. Geometry of Normal Straight Line Parameterization

Each line L is determined by ρ, the distance between its normal and the origin, and θ, the angle the normal forms with the x axis.

One of the major advantages of this Hough space parameterization is computational simplicity. Using an accumulator technique, the normal straight line Hough transform is quickly computed by incrementing points in the Hough parameter space along the sinusoid defined by Eqn (2.3) for each edge point (x,y) within the input scene. When edge points in an input scene are situated along a straight line, the corresponding sinusoids will intersect at a single point causing a maxima at that point within the Hough space (Figure 2.3).

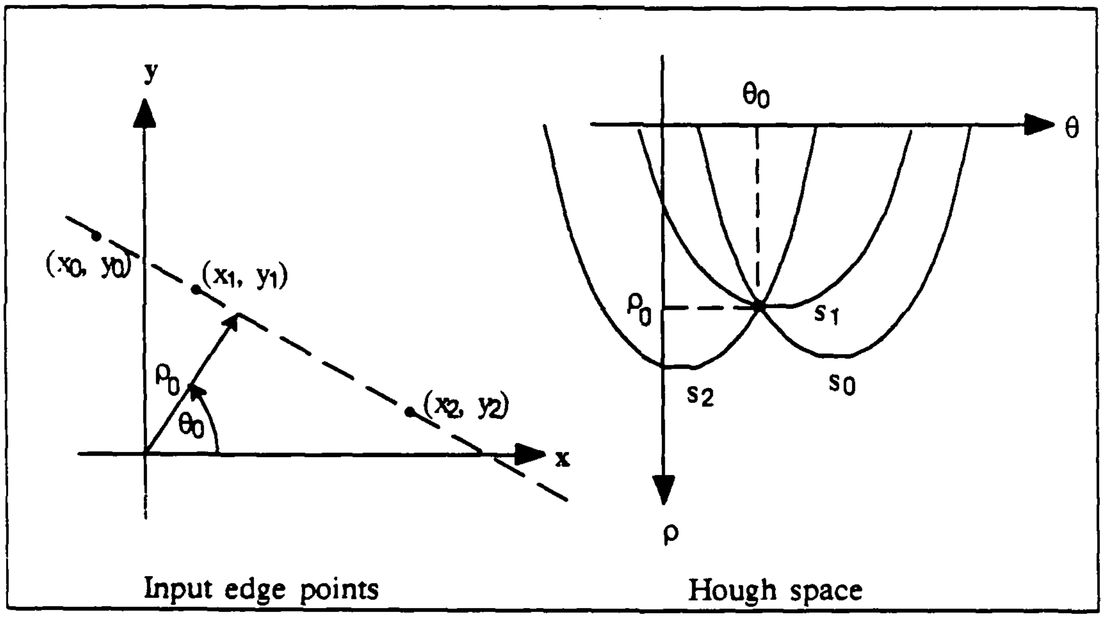


Figure 2.3. Accumulator Implementation of Hough Transform

Each input edge pixel creates a sinusoid in the Hough parameter space. Input points (x_0, y_0 an increased knowledge),(x_1, y_1) and (x_2, y_2) create sinusoids s_0, s_1 and s_2 in the Hough parameter space.

Since $H(\theta, -\rho) = H(\theta + 180, \rho)$, the Hough space is depicted with $\rho \geq 0$ and $0 \leq \theta < 360$. In addition to computational efficiency, work performed by

Casasent–Krishnapuram [8], [9] documented some extremely significant characteristics of the normal straight line Hough space parameterization.

In these articles, Casasent and Krishnapuram studied shifts, rotations and scales of edge curves within input scenes and showed that the corresponding distortions within the Hough parameter space followed well defined rules. The relationship between the Hough transform of an input curve and the Hough transform of its scaled version was shown to be

$$H_s(\theta, \rho) = H(\theta, s\rho) \tag{2.4}$$

Thus, scaling of the input space by s, causes the Hough space to be scaled along the ρ axis [9:304] by the same value.

An object rotated by ϕ in the input space caused the relationship

$$H_r(\theta, \rho) = H(\theta - \phi, \rho) \tag{2.5}$$

where the original Hough space is shifted along θ by ϕ [9:305].

Translations of an object in the input scene such that

$$f_t(x, y) = f(x - a, y - b)$$

creates a sinusoidal distortion

$$H_t = \begin{cases} H(\theta, \rho - t\cos(\theta - \alpha)), & \rho + t\cos(\theta - \alpha) \geq 0 \\ H(\theta - \pi, -\rho - t\cos(\theta - \alpha)), & \rho + t\cos(\theta - \alpha) < 0 \end{cases} \tag{2.6}$$

$$\text{where} \quad t = \sqrt{a^2 + b^2} \quad \text{and} \quad \alpha = \tan^{-1}\left(\frac{b}{a}\right)$$

along the ρ axis of the Hough space [9:306].

The two cases in Eqn (2.6) are merely the result of discarding negative values of ρ and does not point to any inherent discontinuity in the normal straight line parameterization. If the redundant representation of the Hough space where used, the *first case of Eqn* (2.6) would be sufficient to describe the Hough space distortion.

Combining Eqns (2.4), (2.5) and (2.6) gives the generalized characterization for distortion of the Hough parameter space for scales, rotations and shifts (translations) of an object in the input space [9:306]:

$$H' = \begin{cases} H(\theta - \phi, s\{\rho + t\cos(\theta - \alpha)\}), & \rho + t\cos(\theta - \alpha) \leq 0 \\ H(\theta - \phi - \pi, s\{-\rho - t\cos(\theta - \alpha)\}), & \rho + t\cos(\theta - \alpha) < 0 \end{cases} \qquad (2.7)$$

Thus, a search for any two-dimensional object within an input scene can be performed in the Hough parameter space by creating a Hough transform template and using a distort and compare routine to find it's location, rotation and/or scale.

E. *Optical Implementations.* The normal straight line Hough transform can be shown to be a special case of the Radon transform (Appendix A), where

$$H(\theta, \rho) = \iint_{-\infty}^{\infty} f(x, y)\, \delta[\rho - x\cos(\theta) - y\sin(\theta)]\, dx\, dy \qquad (2.8)$$

and

$$f(x, y) = \begin{cases} 1, & (x, y) \text{ is an edge pixel} \\ 0, & (x, y) \text{ is not an edge pixel} \end{cases}$$

Deans [14:96–100] describes a direct relationship between the Radon transform and the Fourier transform. This relationship suggests the Radon transform, and thus the normal straight line Hough transform, can be obtained in real time using some type of optical implementation.

Eichmann and Dong [13] developed a method of optically generating slices of the Hough transform. As shown in Figure 2.4, the input image is placed in plane P_1, a slit placed in plane P_2 passes an angular slice of the Fourier transform of the input. Plane P_3 records $H(\theta_i, \rho)$ for each rotation, θ_i, of the image in plane P_1.

Other optical implementations follow the same form as the Eichmann–Dong method. Steier and Shori [11] employed a similar method. They used a rotating dove prism to effectively rotate an input wavefront and a cylindrical lens to eliminate the slit in plane P_2 of Figure 2.4. Another, somewhat more elaborate, approach was used by Ambs *et al* [12]. They were able to obtain several samples of the the Hough transform through a Fourier transform filter composed of a matrix of holograms. No method, however, has been developed to generate a continuous optical reproduction of the Hough transform.

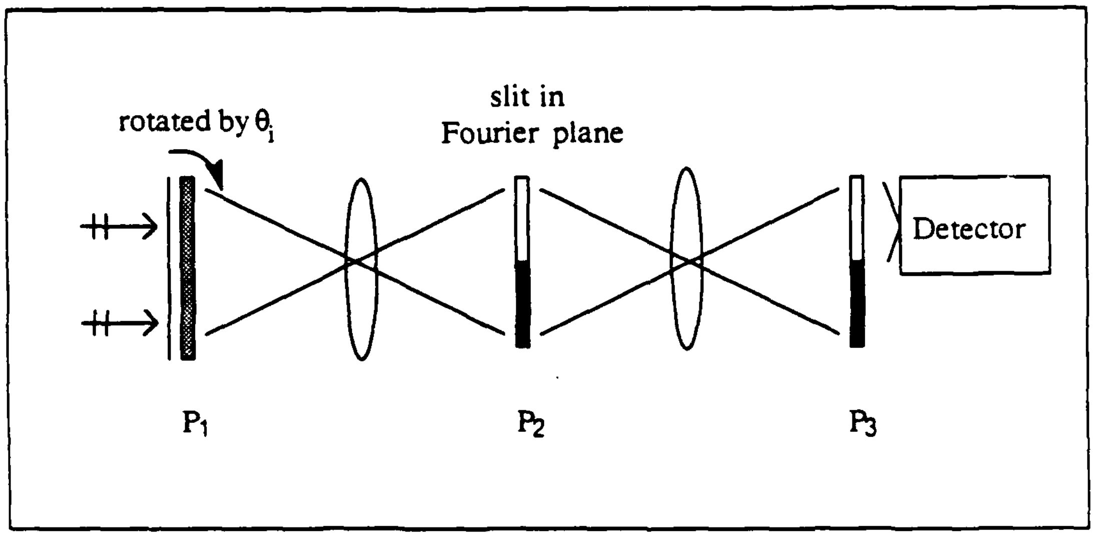


Figure 2.4. Eichmann-Dong Implementation [13:831]

Detector records a separate image for rotation of the input image. Electronic post-processing is required to assemble the final Hough image.

As mentioned in part A of this section, the Hough transform operates on edge images. The next section will discuss methods used to create these edge images.

III. Creation of Edge Images

A. *Source Data.* The edge images used in this thesis were generated from doppler and laser radar digital image data obtained from the Air Force Wright Aeronautical Laboratories (AFWAL) Avionics Laboratory. Computational processing of this data was performed on a VAX 11/780 and an Evans and Sutherland PS340 Graphics Workstation was used for displaying images.

B. *Edging Process.* The block diagram in Figure 3.1 outlines process used to create edge images from the raw doppler and laser radar data.

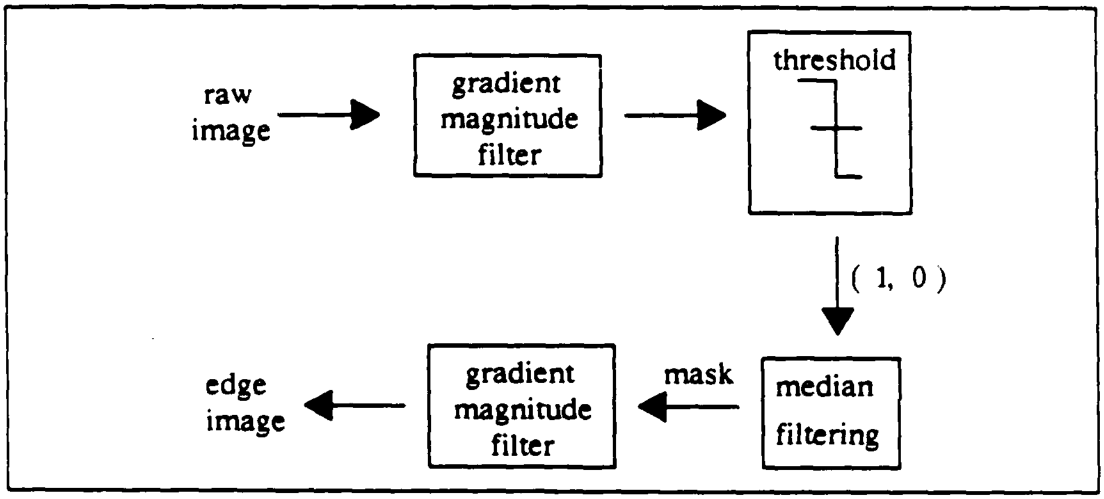


Figure 3.1. Block Diagram of Edging Process

Raw image data was passed through a Tong [20:24-37] gradient magnitude filter. The result was thresholded and binarized. Several stages of median filtering [20:49] were applied to remove unwanted noise. This results in a solid mask of the object. Mask is then sent through another gradient magnitude filter to produce the edge images. A raw doppler image and its corresponding edge image are shown in Figures 3.2 and 3.3. In general, the full Tong algorithm [21] or a similar procedure would be used to segment the mask of the object from an input scene.

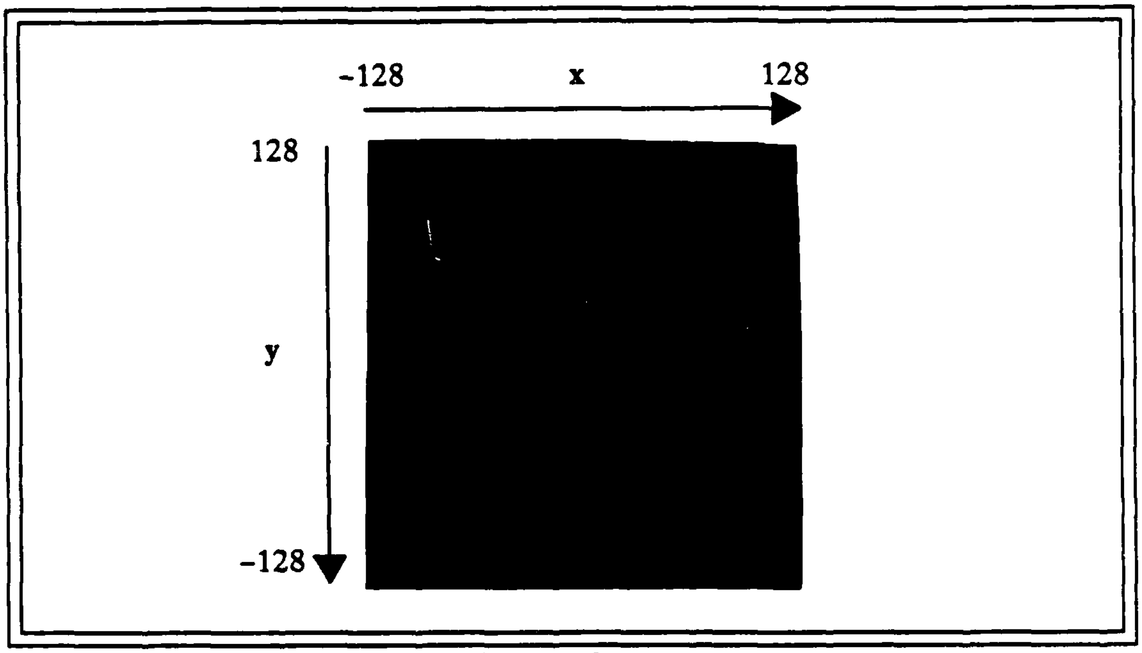


Figure 3.2. Typical Raw Doppler Image

Raw doppler image of a tank obtained from
the AFWAL Avionics Laboratory at Wright-Patterson AFB, OH.

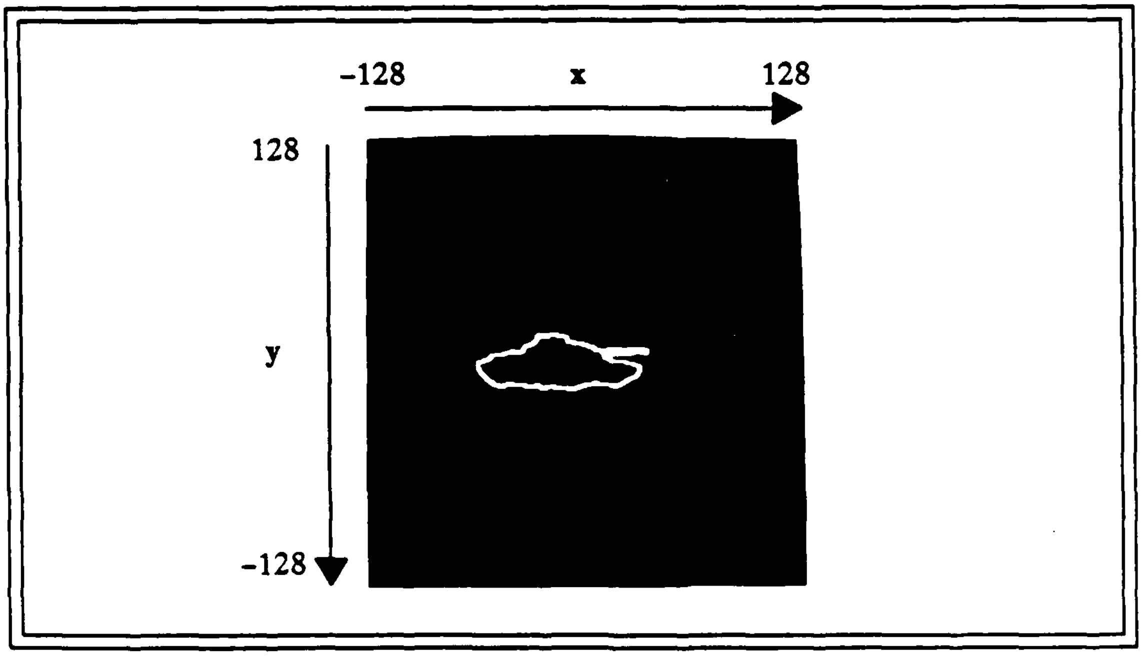


Figure 3.3. Resultant Edge Image

Tank image of Figure 3.2 after edge processing
diagramed in Figure 3.1.

Once a set of edge images has been created, Hough transform processing can begin. The next chapter discusses two methods of producing the Hough transform along with some examples of transformed images to illustrate the effects of distortions of an input object on the Hough transform domain.

IV. Hough Transform Images

Throughout the rest of this thesis, the term Hough transform will apply to the normal straight line Hough transform mentioned in Section II. References to other Hough transform techniques will still be preceded by the specific parameter space identification.

A. *Accumulator Technique.* As mentioned in Section II.D, the Hough transform can be computed rapidly using an accumulator technique. For each edge point, a sinusoid was drawn in Hough space according to Eqn (2.3)

$$\rho = x \cos(\theta) + y \sin(\theta) \tag{2.3}$$

as depicted in Figure 2.3. The Hough transform space with ρ ranging from 0 to 181 (128 multiplied by the square root of 2) and θ ranging from 0 to 359 was created from 256 x 256 pixel edge images.

The Hough transform was first applied to simple shapes to best illustrate some of its interesting qualities. The line in Figure 4.1 creates a local maxima at a single point in the Hough space as shown in Figure 4.2. The ability of the Hough transform to detect different lines of the same slope and also detect both horizontal and vertical lines is displayed in Figures 4.3 and 4.4 by the Hough transform of four line segments arranged in the shape of a box. A centered circle (Figure 4.5) creates maxima along the horizontal line with ρ equal to the radius of the circle (Figure 4.6) as the Hough transform treats small arc segments of the circle as line segments.

The case involving the circle is somewhat interesting since, theorectically, the Hough transform should create a constant level two out to the radius of the

circle, followed by a level one at ρ equal to the radius. This would be expected because the line integration (Eqn 2.8) intersects the circle at two points out to the radius where the line is tangent to the circle.

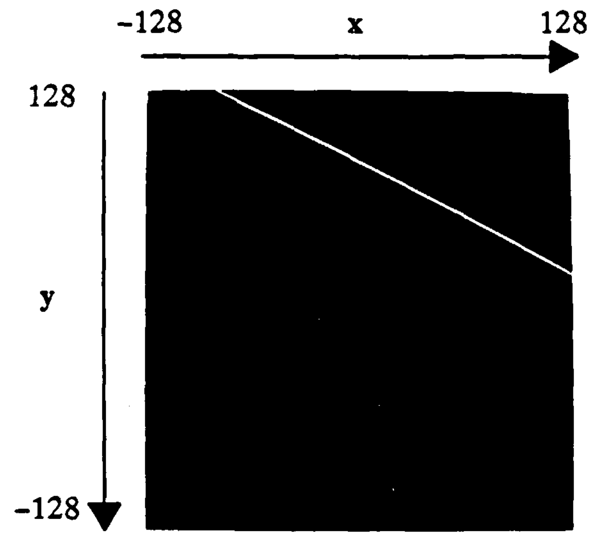


Figure 4.1. Line Edge Image

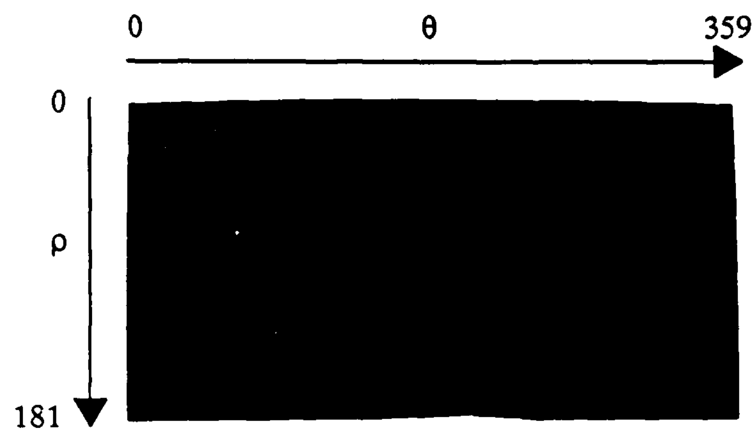


Figure 4.2. Hough Transform of Figure 4.1

The line of Figure 4.1 is described by a local maxima at $\rho = 76$ and $\theta = 63$ in the Hough transform.

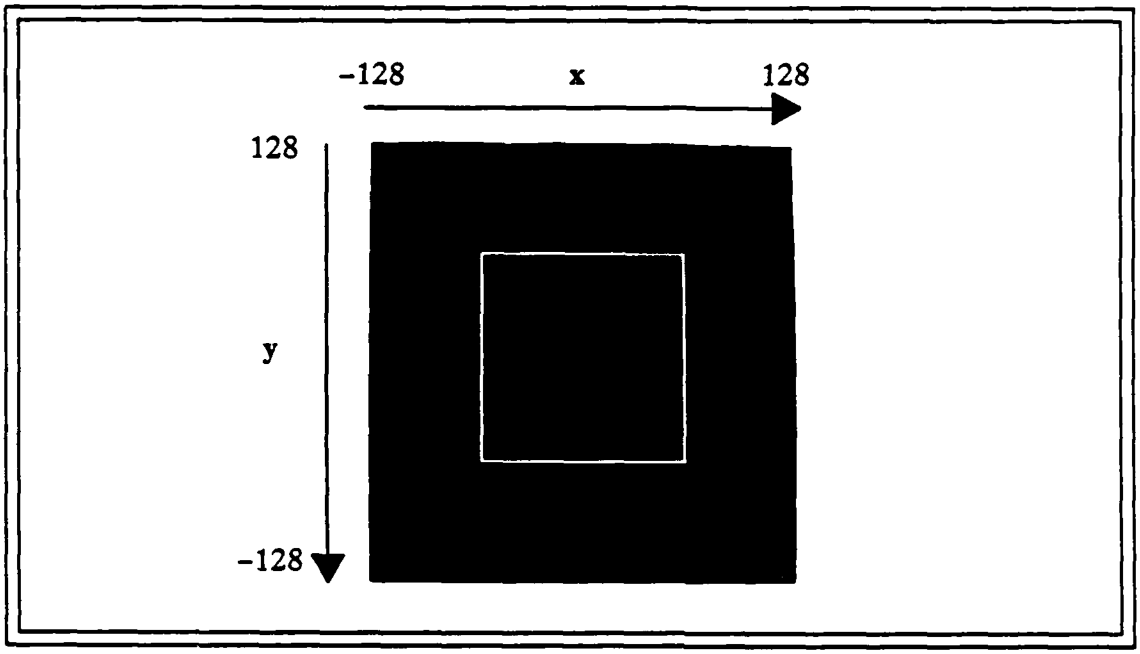


Figure 4.3. Box Edge Image

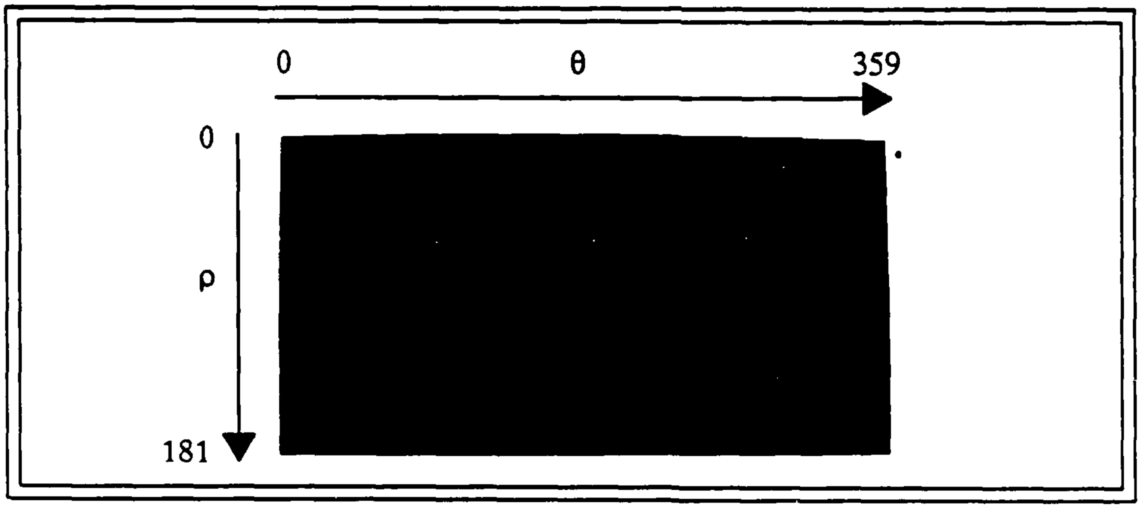


Figure 4.4. Hough Transform of Figure 4.3

Local maxima appear at $\theta = 0°$, $90°$, $180°$ and $270°$ corresponding to each of the straight lines in the original image. Notice the wrap around at $\theta = 359°$ and $\theta = 0°$.

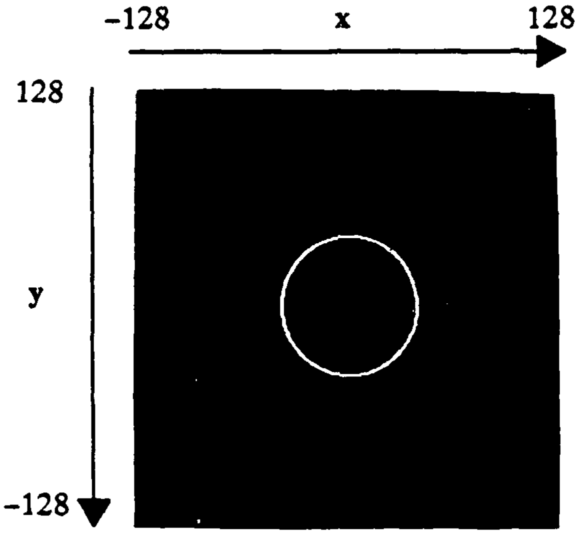


Figure 4.5. Circle Edge Image

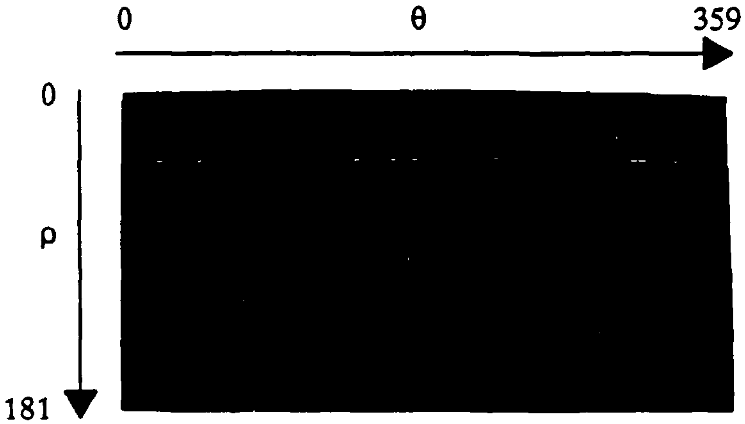


Figure 4.6. Hough Transform of Figure 4.5

Hough transform of a centered circle forms a straight line with ρ = circle radius.

The Hough transform was also applied to the edge images created from raw image data. A typical edge image and its corresponding Hough transform are shown in Figures 4.7 and 4.8.

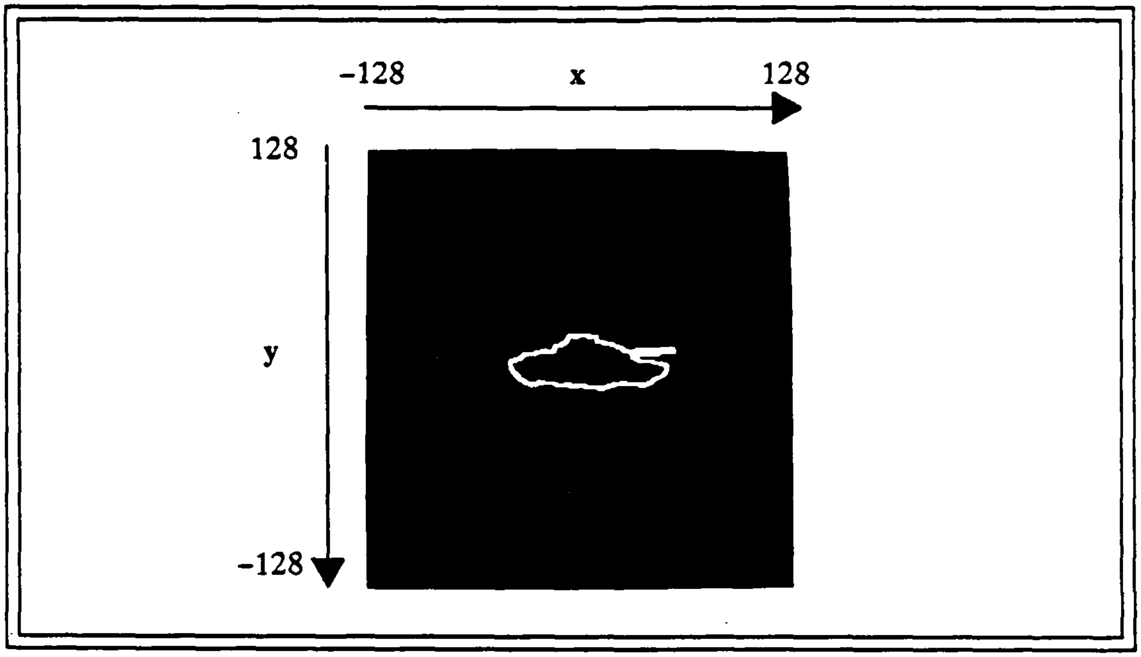


Figure 4.7. Tank Edge Image

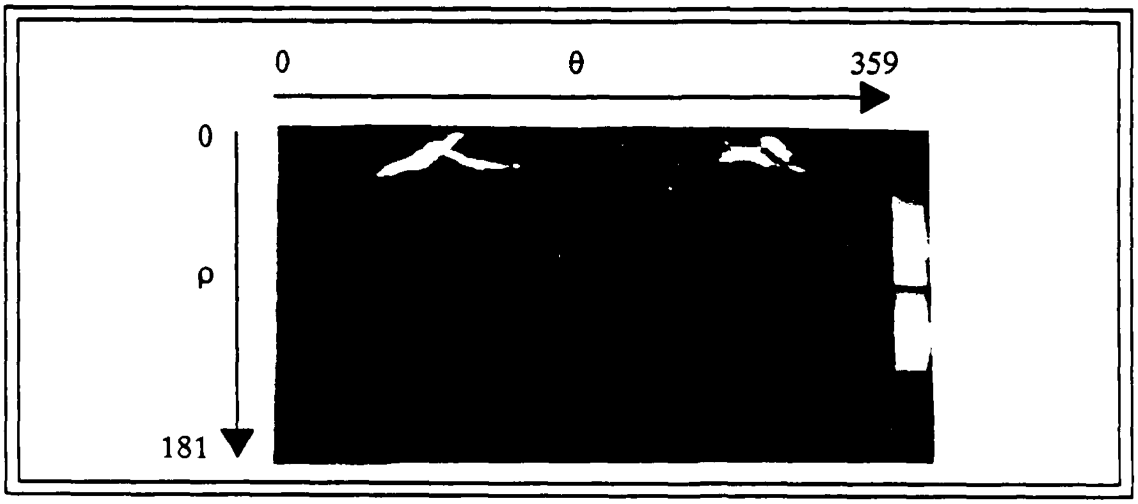


Figure 4.8. Hough Transform of Figure 4.8

The low value artifacts in this color enhanced picture shows the effect of the 'oval' tank shape on the Hough transform. Notice maxima near 90° and 270° where the original tank image approaches a straight line.

An artifact of the accumulator technique is the creation of many points in the Hough space with low accumulator values. This is evident in the color enhanced version of Figure 4.2 displayed in Figure 4.9

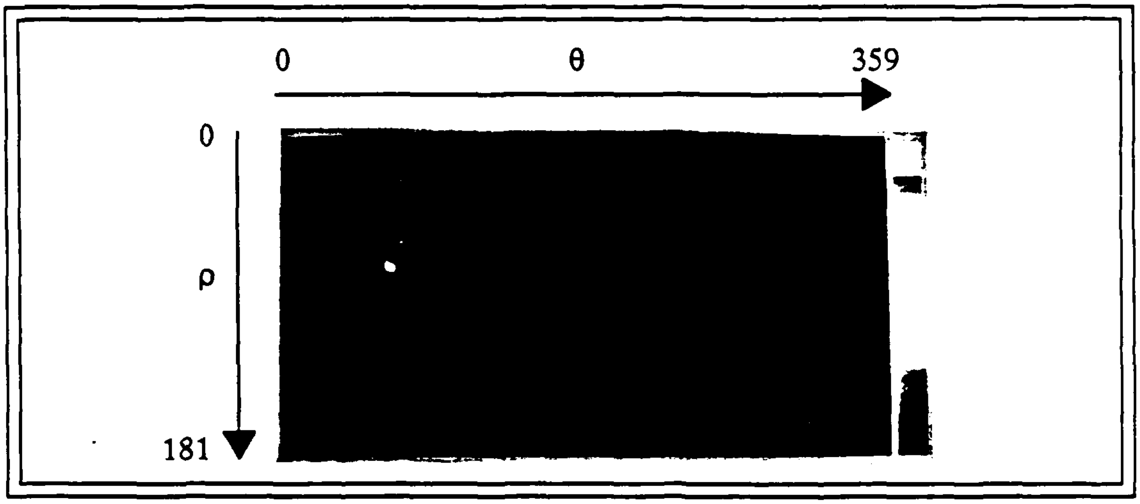


Figure 4.9. Color Enhanced Version of Figure 4.2

and its histogram presented in Figure 4.10.

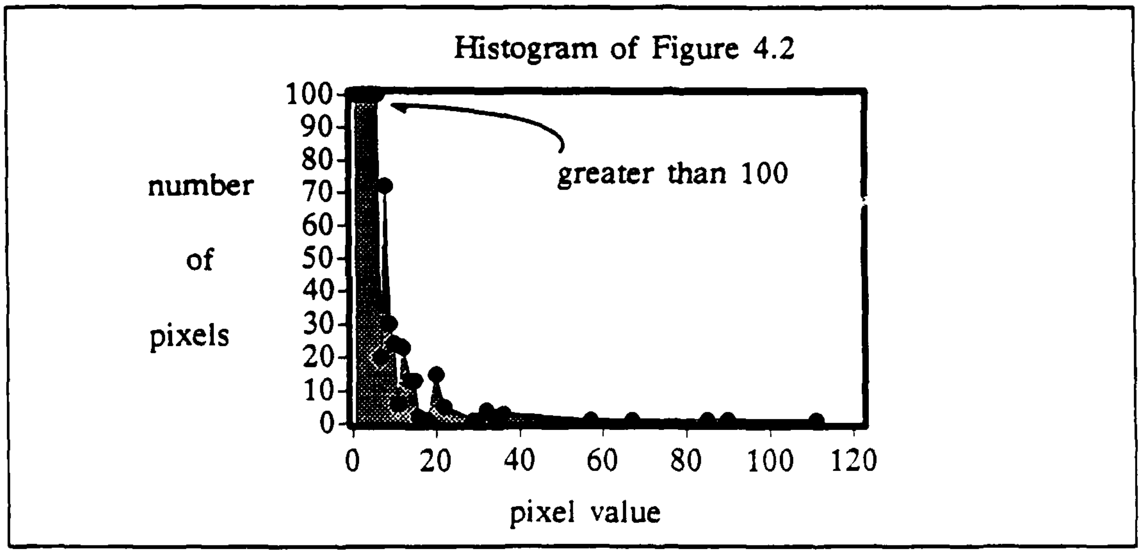


Figure 4.10. Histogram of Figure 4.2

A large number of low pixel value artifacts are created in the accumulator technique. For example, over 16,000 pixels have an accumulator value of 1.

Hough transform images can be thresholded to remove this artifact before images are compared. It should be noted, however, that useful information can be extracted from the artifact itself. The finite extent of this Hough space artifact can help determine the finite extent of the curve in the edge image.

B. *Fourier Method.* The Hough transform, when expressed in integral form (Eqn 2.8), is a special case of the two-dimensional Radon transform (Appendix A) were the input, f(x,y), is a binary function.

$$H(\theta, \rho) = \iint_{-\infty}^{\infty} f(x, y)\ \delta[\rho - x\cos(\theta) - y\sin(\theta)]\ dx\ dy \qquad (2.8)$$

This integral relationship can be implemented through use of Fourier transforms [14:96-100]. The block diagram of Figure 4.11 outlines the procedure used in this thesis to generate this Hough transform implementation.

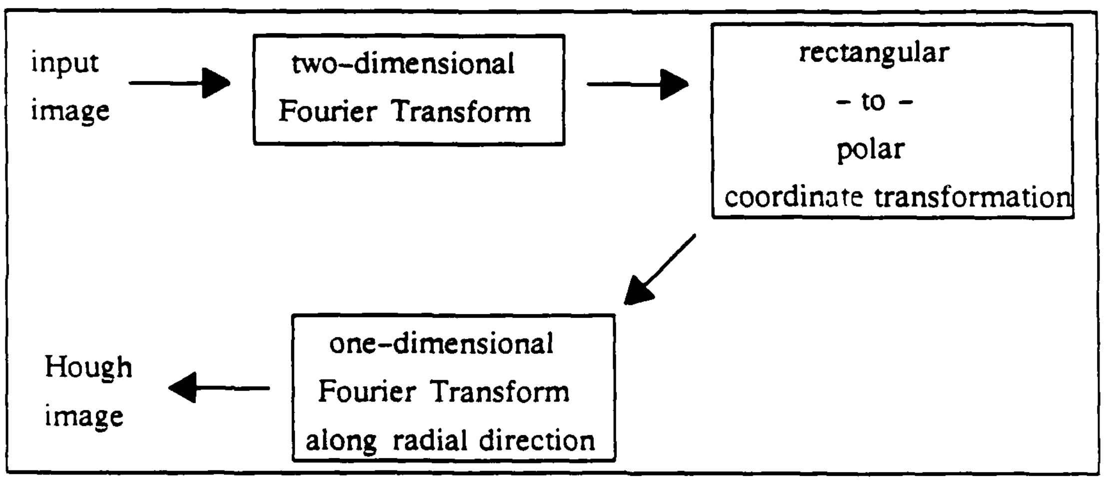


Figure 4.11. Fourier Transform Implementation of the Hough Transform

A two-dimensional Fourier transform is applied to the image as defined in Eqn (4.1).

$$F(u, v) = \iint_{-\infty}^{\infty} f(x, y)\, e^{-j2\pi(xu + yv)}\, dx\, dy \qquad (4.2)$$

A Cartesian-to-polar change of coordinates is performed on the spatial frequency variables u and v such that

$$r = \sqrt{u^2 + v^2} \quad \text{and} \quad \theta = \tan^{-1}(v / u)$$

thus,

$$F(\theta, r) = \iint_{-\infty}^{\infty} f(x, y)\, e^{-j2\pi[\,x r \cos(\theta) + y r \sin(\theta)\,]}\, dx\, dy \qquad (4.3)$$

Finally, by performing a one-dimensional inverse Fourier transform along the radial coordinate of Eqn (4.3),

$$H(\theta, \rho) = \int_{0}^{\infty} \iint_{-\infty}^{\infty} f(x, y)\, e^{-j2\pi[\,x r \cos(\theta) + y r \sin(\theta)]}\, dx\, dy\, e^{j2\pi r \rho}\, dr \qquad (4.4)$$

and rearranging terms,

$$H(\theta, \rho) = \iint_{-\infty}^{\infty} \int_{0}^{\infty} f(x, y)\, e^{-j2\pi r\,[\,\rho - x\cos(\theta) - y\sin(\theta)\,]}\, dr\, dx\, dy \qquad (4.5)$$

the inner integral simplifies to a Dirac delta function and Eqn (2.8) results.

Implementing this procedure with discrete Fourier transforms, the Hough transforms of edge images in Figures 4.3, 4.5 and 4.7 are shown in Figures 4.12 through 4.14.

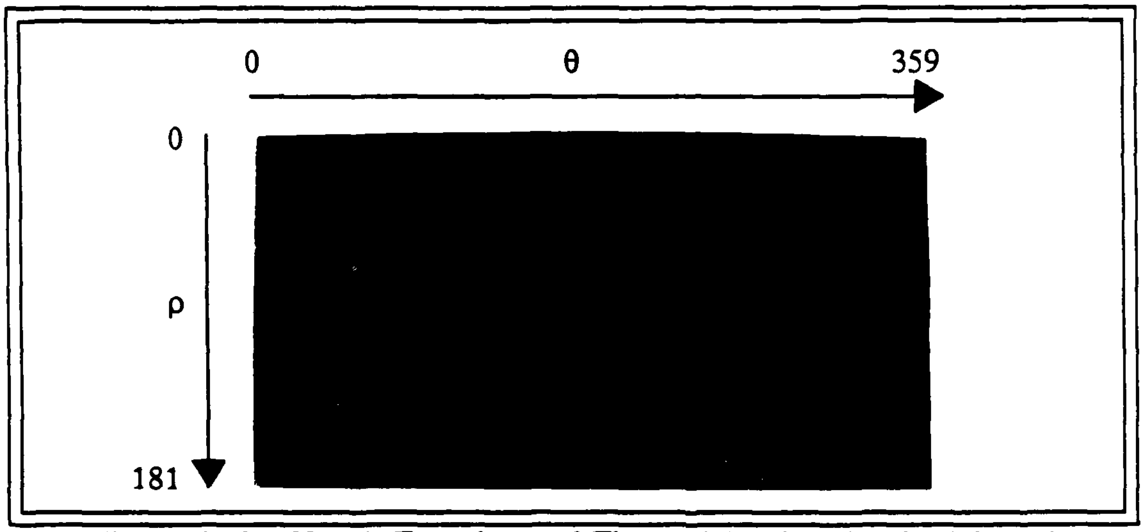


Figure 4.12. Hough Transform of Figure 4.3 using Fourier Method

Hough transform of the box edge image in the Figure 4.3 obtained by using discrete Fourier transforms. Maxima appear at the same positions as Figure 4.4.

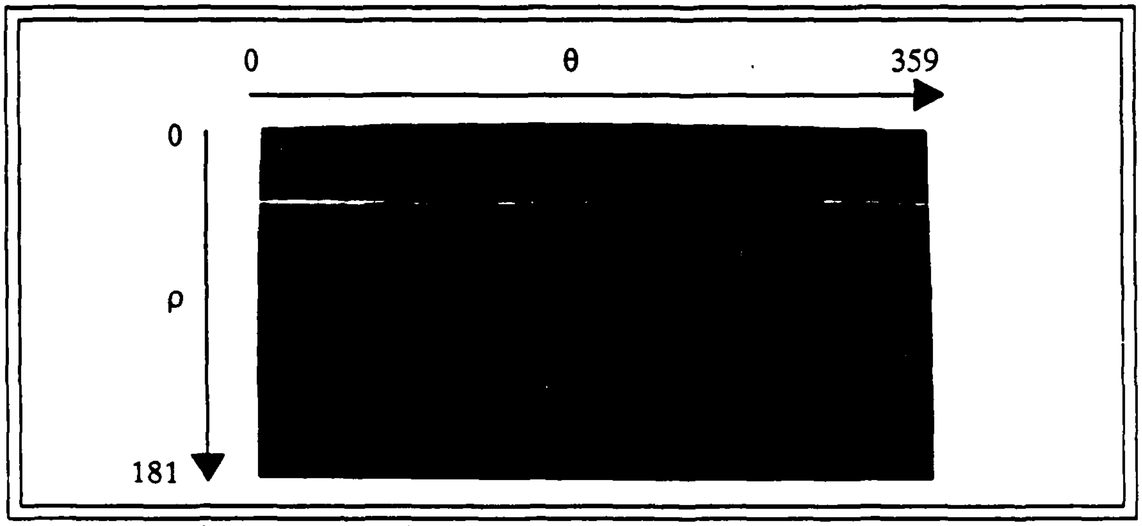


Figure 4.13. Hough Transform of Figure 4.5 using Fourier Method

Straight line in same position as Figure 4.6.

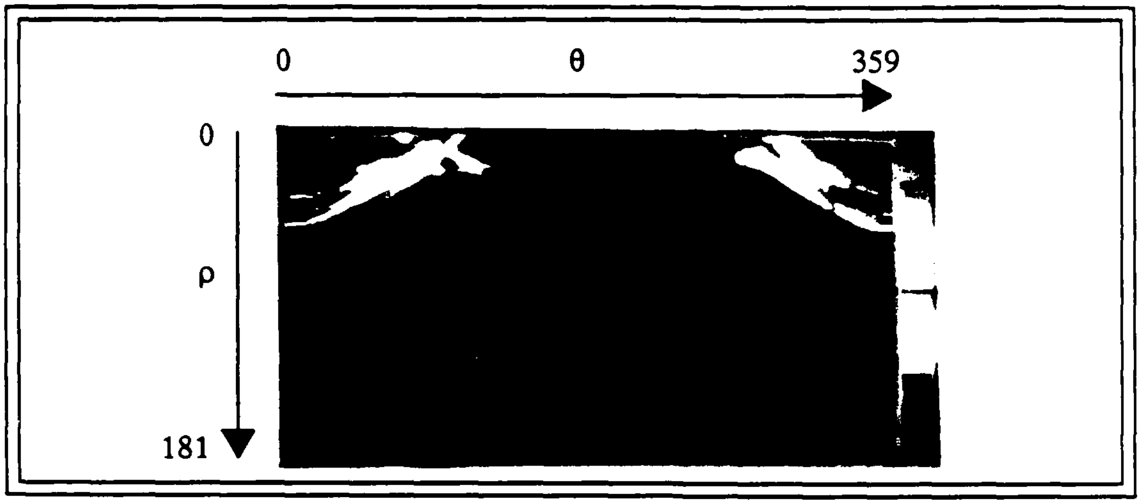


Figure 4.14. Hough Transform of Figure 4.7 using Fourier Method

Notice that the outlines created by the artifacts in Figure 4.7 are present are also present when Fourier transforms are used to obtain the Hough transform.

Since Fourier transforms are invertable, the original edge images were retrieved from the Hough transforms by using discrete Fourier transforms and a polar-to-rectangular coordinate transformation to perform the inverse of Figure 4.11.

The chapter outlined the distortion characteristics of the Hough transform due to shifts, rotations and translations of an input object. Since these distortion characteristics follow well defined rules, they can be used to estimate those input object parameters in the Hough space. There are some interesting advantages to this technique and they will be discussed in the next chapter.

V. Determining Scale, Rotation, Shift and Location

A. *Distortion Characteristics.* The distortion characteristics of the Hough transform were described mathematically by Eqns. (2.4) through (2.7). Experimental results verify these equations. The effects of scales and shifts (translations) of an input curve on the Hough transform are best understood by examining the Hough transform of a circle. A centered circle of 40 pixel radius and its Hough transform were shown in Figures 4.5 and 4.6. The effect due to scale is evident when examining the Hough transform of the circle of 25 pixel radius (Figure 5.1) displayed in Figure 5.2. The effects of small and large shifts of the circle in Figure 4.5 (shifts are 45^{c} with respect to the x axis) from center are presented in Figures 5.3 through 5.6. Notice that when the sinusoidal shifting of the ρ axis produces negative values, they are shifted in theta by 180 degrees. This accounts for band of theta values containing zero energy in Figure 5.6.

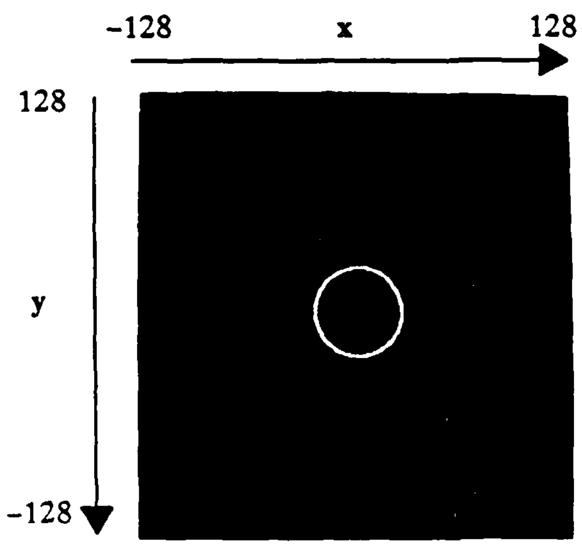


Figure 5.1: Circle of 25 pixel radius

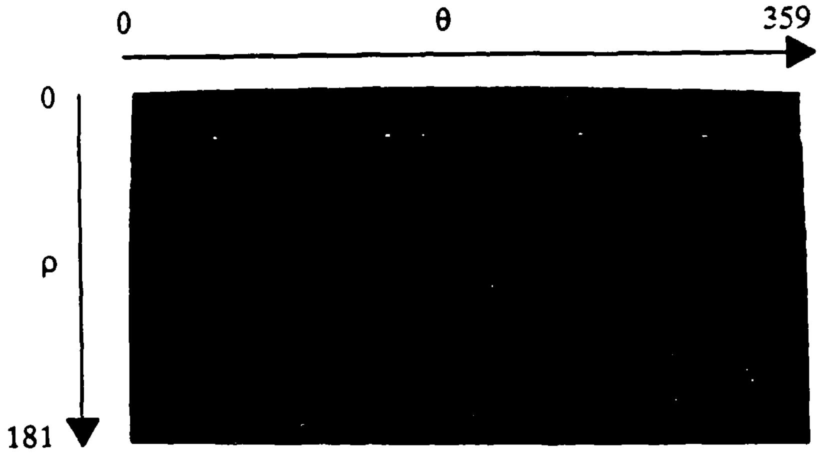


Figure 5.2: Hough Transform of Figure 5.1

Notice the scaling of ρ when compared to the Hough transform of the 40 pixel circle of Figure 4.6.

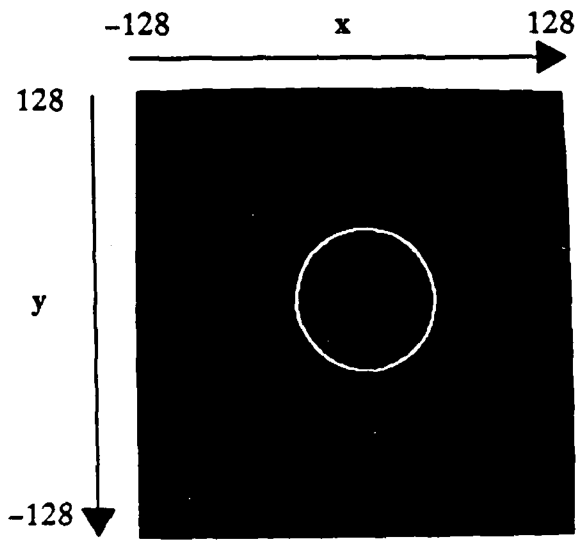


Figure 5.3: Circle of with Small Shift

The 40 pixel circle of Figure 3.3 is shifted by 5 pixels. The shift angle is 45° to the x axis.

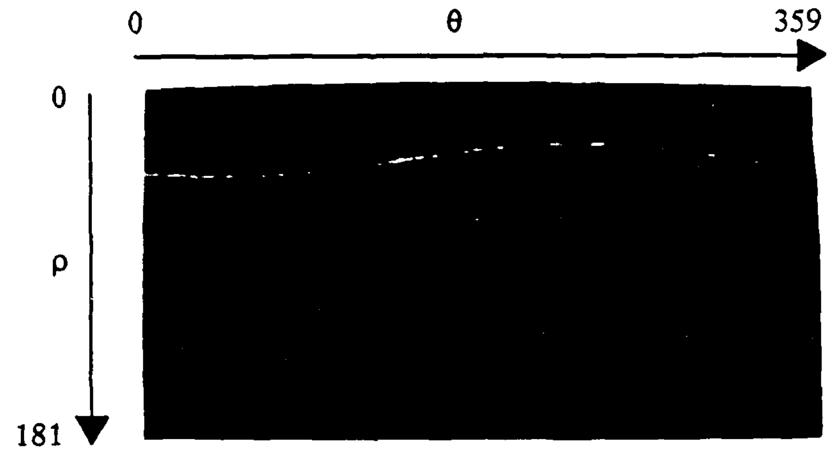


Figure 5.4: Hough Transform of Figure 5.3

The Hough transform of a shifted 40 pixel circle is distorted in a sinusoidal fashion along the ρ axis.

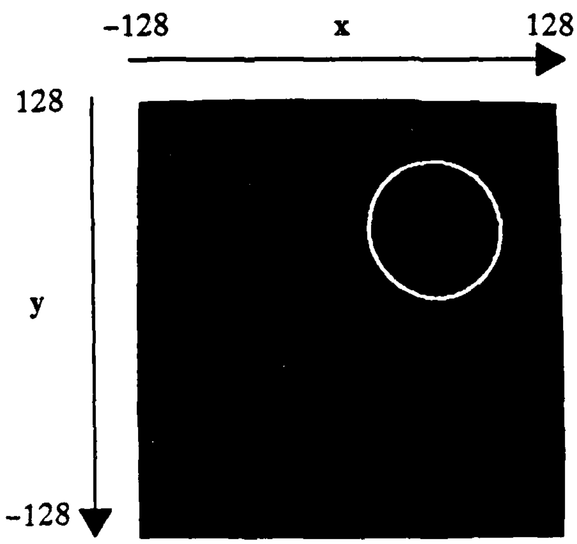


Figure 5.5: Circle with Large Shift

The 40 pixel circle of Figure 3.3 is shifted by 60 pixels. The shift angle is 45° to the x axis.

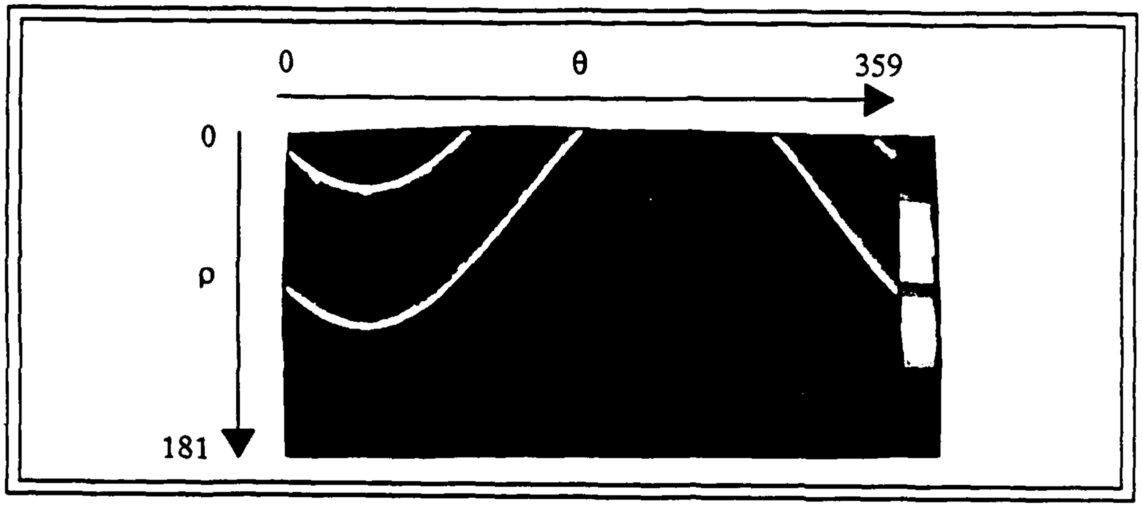


Figure 5.6: Hough Transform of Figure 5.5

Bands of zero energy, in θ, created by areas of the Hough transform shifted by 180° due the creation, and consequent shifting, of negative ρ values.

B. *Parameter Estimation Process.* A distort and compare routine was developed to estimate, in the Hough transform domain, scales, rotations and shifts of a desired object within an input scene. The block diagram of Figure 5.7 outlines the process. Both input and template Hough transforms are thresholded to minimize computations. The estimator is provides with search ranges and increment values for scales, rotations and shifts. The template is distorted according to Eqn 2.7 for each value in the range. The Hough transform of the input is then compared to the template. Scale, rotation and shift values corresponding to the best degree of match (correlation) between input and template are saved and used to compute the location of the object in the input scene.

Various methods can be used to limit the search. One is to observe the zero energy portions of the unthresholded input Hough space and omit shift values within this range. This, of course, is only possible when the Hough space is

determined by the accumulator technique. Another method is to provide the estimator with variable increments. The initial search could be started with coarse increment values. The increment values could then be decreased as the degree of match (correlation) increases. A final method would be to initially center the edge images and thus only search for scales and rotations of the template.

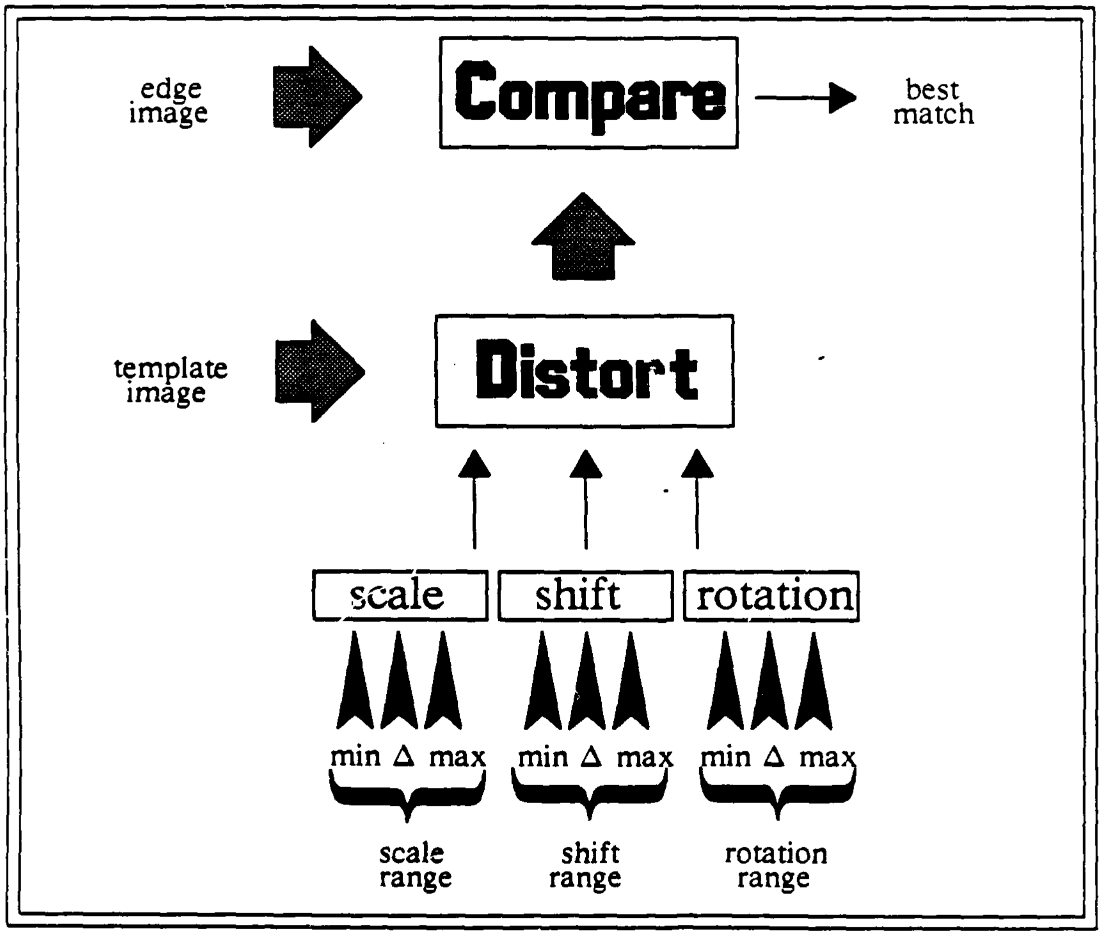


Figure 5.7. Estimation Process

The thresholded edge Hough transform image is compared to a template Hough transform image that has been distorted to simulate shifts, rotations and scales in the input image space.

Tables 5.1 through 5.3 list the performance of the process in estimating locations, rotations and scales respectively.

Table 5.1: Location Estimation			
actual		estimated	
X	Y	X	Y
5	15	5	15
24	25	24	25
36	-50	36	-50
45	35	45	35
40	60	40	60
-60	60	-60	60

The Hough transform template for Table 5.1 developed using a centered version of the edge image shown in Figure 3.3. The edge image was shifted in the scene by the values in the right hand side of the table. The Hough transform of the shifted image estimated using the process diagramed in Figure 5.7.

The one degree resolution of the Hough space was able to determine location to the nearest pixel in all objects of reasonable size. Location estimates were developed by shifting a template edge image in an input scene and applying the Hough transform to create a new input to the estimator.

Table 5.2: Rotation Estimation	
actual	estimated
30	30
90	90
120	120
200	200
260	260
300	300
335	335

Table 5.2 shows Hough space estimations of the same tank image rotated in the input (edge) space by the values shown.

Table 5.3: Scale Estimation		
tank	actual	estimated
d3033	1.0	1.0
d3108	0.7	0.7
d3042	0.4	0.4
d3074	0.3	0.2

Table 5.3 shows scale estimates of different tank images using d3033 as a template.

Rotation estimates were also within the increment resolution of the search. The accuracy of the location and rotation estimates indicates that a Hough space of less resolution would be sufficient to characterize most input scenes. The scale estimates used similar tank edges of differing sizes. Actual size of these cases were determined by length and width comparisons with the template. The scale estimator was unable to accurately determine scales below approximately 0.4 due to collapse of the Hough space about $\rho = 0$.

This chapter has demonstrated how the distortion characteristics of the Hough transform due to rotations, scales and shifts of an input object can be used to estimate these parameters in the Hough space. The following chapter contains an overall discussion of methods used in this thesis and there relative performance.

VI. Discussion

A. *Accumulator Method.* The accumulator method provided an efficient means of producing the Hough transform. The low-level noise artifacts present a major limitation to employing the transform to more than one segmented edge object at a time. However, by applying the transform to each area and thresholding them separately to remove the artifacts, the method can be used to characterize differing image outlines. In well segmented images, the artifacts actually proved to be useful in reducing the range of the search in the parameter estimation process. Segmentation noise has a non-linear effect on the Hough transform. Stray pixels present in the outer portions of the image produce more noise in the transform than those toward the center since the sinusoid it produces is weighted by the distance of the pixel from the origin. Thus, adequate median filtering is essential in the initial segmentation process before application of the Hough transform.

B. *Parameter Estimation.* When artifact information can be employed, or when the input image is centered initially to narrow the range of the search, the parameter estimation method can distinguish between differing images with reasonable efficiency. Size and rotation information are also extractable in the estimation process. Rotation increments near 6 degrees and size increments of approximately 0.1 proved quite adequate for estimating locations of a template among multiple objects in an image. Rotation and shift were determined exactly (or to the resolution of the search increment) while scale estimates proved accurate when the input image was at least 0.4 of the template.

C. *Fourier Method.* Use of discrete Fourier transforms and a coordinate transformation to produce the Hough transform correlated highly with the accumulator method. The 'butterfly' patterns of line segments, as well as the more intricate variations in complex shapes matched those produced by the accumulator method. The Hough transform of a circular object was highlighted in Section IV.A noting the maximum created in the Hough space at ρ equal to the radius of the circle when the accumulator method was used. Interestingly, this maxima is also present in the Fourier created Hough space. Thus, the Fourier method treats small arc segments in the same fashion as the accumulator method. Notice that if a two-dimensional delta function is inserted into Eqn (2.5), or Eqn (4.8), the result is a line mass distributed along a sinusoid. In both the continuous and discrete (accumulator) implementations, each point in the input contributes to a sinusoid in the Hough space. Therefore, what is commonly referred to as the accumulator method should, in fact, be called the discrete Hough transform. The next chapter outlines some attempts to achieve the continuous representation optically which, unfortunately, proved unsuccessful.

VII. Suggested Optical Implementation.

A. *Flawed Approach.* The effort to improve upon current optical implementation methods produced no useful results. The major obstacle to a continuous real-time optical reproduction of the Hough transform is the rectangular-to-polar coordinate transformation required on the complex field in the image Fourier transform plane. One method considered made use of computer-generated interferograms (Appendix C) to implement the coordinate transformation. The computer-generated interferograms, unfortunately, are effective only on planar wavefronts.

B. *New Information.* A recent article by Jensen *et al* [18] suggests a variation on the CGH method. Jensen recorded the phase function of the [x, y] - to - [θ. $\ln\sqrt{(x^2 + y^2)}$] Fourier transform coordinate transform (see Appendix B) on dichromated gelatin. With their phase filter, they effected the coordinate transformation on a complex wavefront. Jensen *et al* claims that derivation of Eqn (B.8) will hold for the Fourier transform of a real input and has published data on performing the coordinate transformation in the Fourier transform plane. They state that as long as the variations in the input are small compared to the variations in the phase filter, the coordinate transformation can be performed. If this is indeed the case, the following optical scheme could be implemented,

C. *Suggested Approach.* By substituting Jenson phase filters, or possibly one-bit phase filters using chemically etched glass, for the computer-generated filters, a possible approach for generating a continuous Hough (Radon) transform is suggested. In the implementation, shown in Figure 7.1, the Fourier transform of

the input is multiplied by the phase function of Eqn (B.8) in plane P_2. This wavefront is Fourier transformed to produce the [θ, ln r] transformation. This wavefront is multiplied by the one-dimensional exponential phase filter

$$\phi_e(\theta, l) = \exp(l) \tag{7.1}$$

This is again Fourier transformed to produce the exponential shifting of the ln r coordinate. Finally, a cylindrical lens is suggested to do the final one-dimensional Fourier transformation.

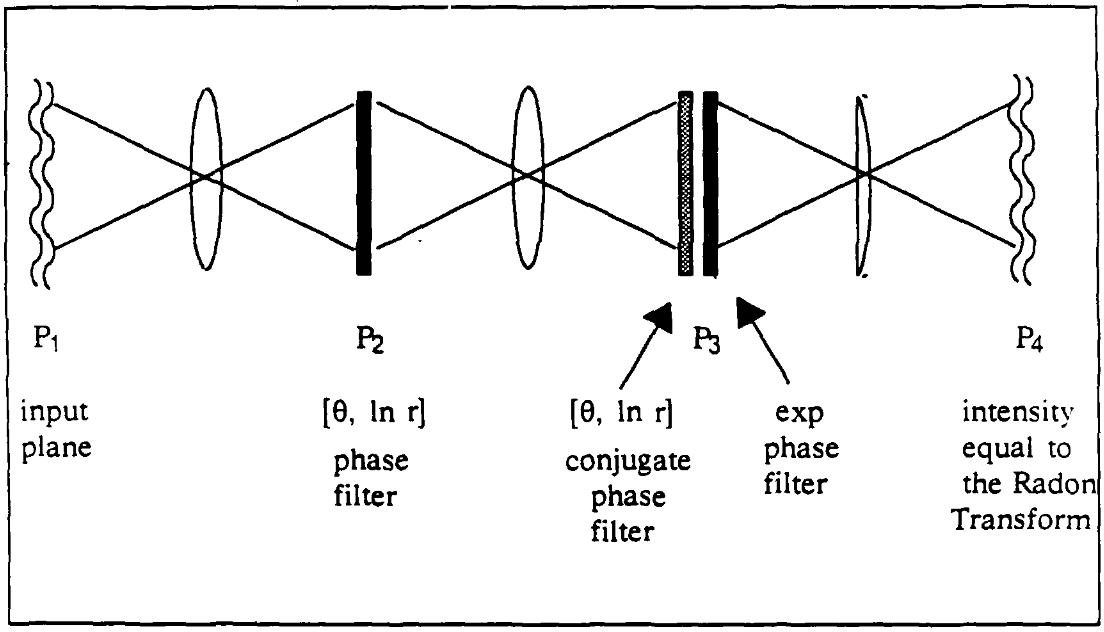


Figure 7.1. Suggested Optical Implementation

D. *Inverse Transform.* Since the method is continuous, it can be reversed to generate the inverse Hough (Radon) transform. Figure 7.2 presents the inverse implementation. The logarithmic one-dimensional coordinate transform phase

filter, Eqn (7.3), is place in P_1 and

$$\phi_1(\theta, r) = \ln(r) \tag{7.3}$$

the phase filter

$$\phi(\theta, l) = \exp(l)\sin(\theta) \tag{7.4}$$

used to produce the [θ, l]-to-[x, y] coordinate transformation. This phase function satisfies the four partial differential equations of Appendix C.

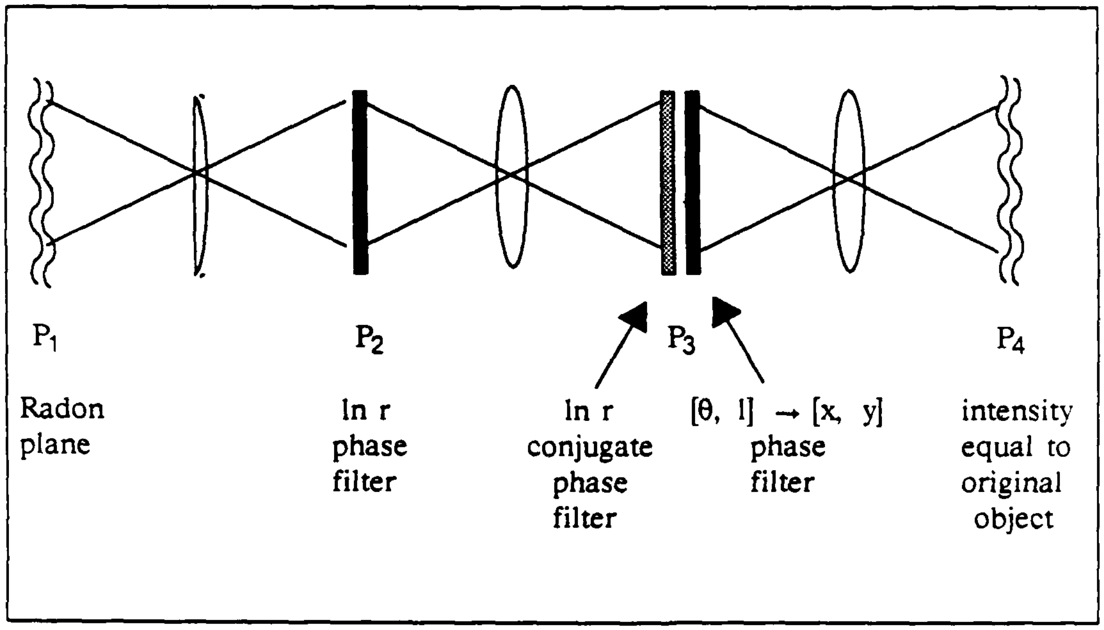


Figure 7.2. Suggested Inverse Optical Hough Transform

The conjugate phase filters shown in Figure 7.1 and 7.2 are introduced to cancel the phase of the previous filter which is reproduced in the back focal plane of the

lens (Appendix B).

This optical scheme is presented as a suggestion for further research. Its feasibility depends highly on the accuracy of the Jensen *et al* development. This point is reiterated in the conclusion to this thesis which follows in next chapter.

VIII. Conclusion

This thesis has tested the feature extraction properties of the Hough transform on simple images and real data. Experimental results have demonstrated that Hough transform techniques to determine shifts, rotations and scales of an input object in the Hough space can be easily implemented through a distort-and-compare process. The accuracy in determining these distortion parameters were primarily influenced by the resolution of the search increments and of the generated Hough space. When the accumulator technique is used, information to reduce the range of the search can be extracted from low pixel value artifacts in the Hough transform of an input edge image.

A continuous implementation of the Hough transform using Fourier transforms was implemented successfully. This points favorably to a possible continuous optical implementation. The main obstacle is producing the necessary rectangular-to-polar coordinate transformation.

Attempts to produce a continuous optical implementation of the Hough transform using computer-generated interferogram coordinate transformations proved unsuccessful. In order for this approach to succeed, the complex field requiring coordinate transformation must be separated and processed individually. A recent article by Jensen *et al* may provide an answer to this problem of a continuous implementation. The assumptions made in his coordinate transformation derivation (Appendix B) may be too restrictive for this application. Thus, it is only presented as a suggestion for further research.

Appendix A: Radon Transform

This appendix is limited to discussion of the two-dimensional Radon transform. An extension to higher orders can be found in Deans [14].

1. *General Definition.* The Radon transform is formed by mapping the projection of some function along all possible lines in a plane. Thus, the line integral of Eqn (A.1) describes the general Radon transform.

$$R(\theta, \rho) = \oint_L f(x, y)\, dl \qquad (A.1)$$

Thus, the first restriction is that this integral must exist. In his paper, *published in 1917* [14:204-217] Radon *showed that the transform must* exist if the following conditions are meet:

a) f(x,y) is continuous

b) the integral of Eqn (A.2) converges

$$\int_{-\infty}^{\infty} \left| \frac{f(x, y)}{\sqrt{(x2 + y2)}} \right| dx\, dy \qquad (A.2)$$

c) and, for in the plane the limit in Eqn (A.3) holds

$$\lim_{r \to \infty} \frac{1}{2\pi} \int_0^{2\pi} f(x + r\cos(\theta), y + r\sin(\theta))\, d\theta \qquad (A.3)$$

These conditions can be satisfied for most functions. For instance, if the function is bounded, the last two restrictions will be satisfied. With discontinuous

function, the remaining condition can still be satisfied by integrating over continuous segments. Starting with the parameterization of Eqn (2.3), and superimposing a new coordinate system (ρ, s) over Figure 2.2 where L is perpendicular to the ρ axis (Figure A.1),

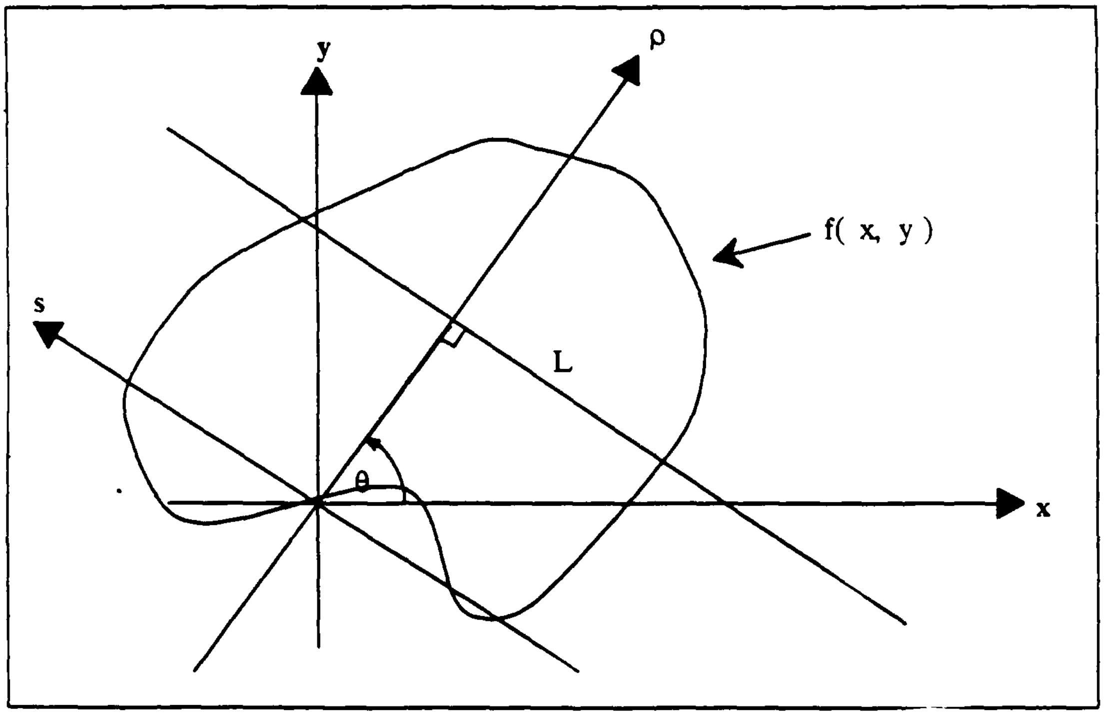


Figure A.1. Line Through f(x, y) [14:57]

each point along L is determined by

$$x = \rho \cos(\theta) - s \sin(\theta)$$
$$y = \rho \sin(\theta) + s \cos(\theta) \tag{A.4}$$

Thus, Eqn (A.1) is explicitly defined as

$$R(\theta, \rho) = \int_{-\infty}^{\infty} f(\rho\cos(\theta) - s\sin(\theta),\ \rho\sin(\theta) + s\cos(\theta))\, ds \qquad (A.5)$$

Alternatively, the Dirac delta function can be employed to select each possible line L of Figure A.1 and the more familiar Eqn (A.6) results.

$$R(\theta, \rho) = \int_{-\infty}^{\infty} f(x, y)\, \delta[\rho - x\cos(\theta) - y\sin(\theta)]\, dx\, dy \qquad (A.6)$$

Appendix B: Coordinate Transformations

Information in this appendix was extracted from Casasent-Psaltis [16], Casasent *et al* [15], Born-Wolf [22], Jenson *et al* [19] and Bryngdahl [18].

1. *Fourier Transform Approach.* Casasent and Psaltis [16] introduced the concept of Fourier transform coordinate transformations. The approach states that an input function, f(x, y), can be multiplied by an appropriately selected phase filter function to produce an output function whose intensity is 'proportional' to some desired f(u, v) where u and v are functions of x and y. This is stated mathematically in Eqn (B.1) and optically implemented in Figure B.1.

$$\iint_{-\infty}^{\infty} f(x, y) \exp\left\{ j \frac{2\pi \phi(x, y)}{\lambda f_L} \right\} \exp\left\{ -j \frac{2\pi (xu + yv)}{\lambda f_L} \right\} dx\, dy \tag{B.1}$$

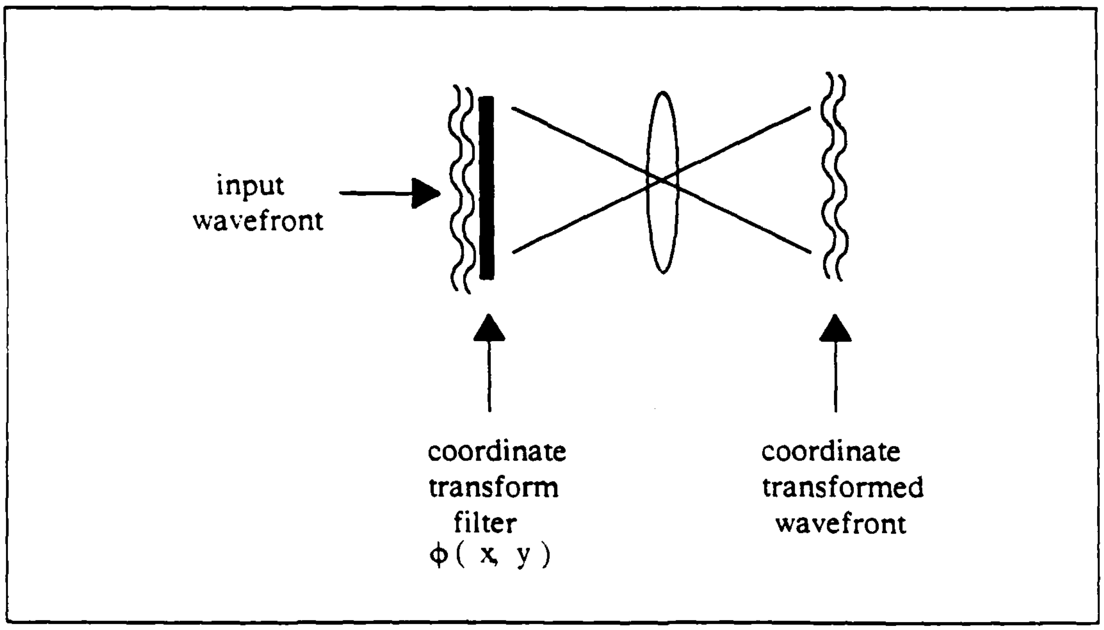


Figure B.1. Optical Implementation of Coordinate Transform Filter

In order to determine the limitations on choosing a $\phi(x, y)$, a solution to Eqn (B.1) must be derived. Methods for solving this double integral are provided by Born–Wolf [22] and Jenson *et al* [16]. Although outlined quite differently, both methods follow the same reasoning. Therefore, only the method published in Born–Wolf [22] will be broached.

The method discussed in Appendix III of Born and Wolf [22] used an asymtotic approximation to solve double integrals of the form

$$\int_{-\infty}^{\infty} f(x, y) \exp\{ j k h(x, y) \} \, dx \, dy \qquad \text{(B.2)}$$

This approach limits $h(x, y)$ to a real function. In general terms, the approach holds that "contributions to the asymtotic expansion (of $h(x, v)$) come only from regions in the vicinity of certain critical points." Three types of 'critical points' are discussed; however, only the case called 'a critical point of the first kind' pertains to this application. At each critical point where,

$$\frac{\delta h}{\delta x} = \frac{\delta h}{\delta y} = 0 \qquad \text{(B.3)}$$

provide a necessary condition for choosing $\phi(x, y)$. Since

$$h(x, y) = \frac{k [\phi(x, y) - x u - y v]}{\lambda f_L} \qquad \text{(B.4)}$$

Eqns (B.5) must hold.

$$\frac{\delta\,\phi(x,y)}{\delta x} = \frac{k}{f_L}\,u\,, \qquad \frac{\delta\,\phi(x,y)}{\delta y} = \frac{k}{f_L} \tag{B.5}$$

Once h is expanded about a critical point (x_0, y_0) and approximated to the first and second terms, the solution to Eqn (B.2) is given by

$$\frac{j\,2\pi\,\sigma}{\sqrt{|\phi_{xx}\,\phi_{yy} - \phi_{xy}^2|}}\; f(x_0, y_0)\,\frac{f_L}{k}\, e^{\,jk\,\phi(x_0,\,y_0)\,/\,f_L} \tag{B.6}$$

where,

$$\phi_{xx} = \frac{\delta^2\phi}{\delta x^2}, \quad \phi_{yy} = \frac{\delta^2\phi}{\delta y^2}, \quad \phi_{xy} = \frac{\delta^2\phi}{\delta x\,\delta y}, \quad k = \frac{2\pi}{\lambda}$$

and

$$\sigma = \begin{cases} 1 & \phi_{xx}\,\phi_{yy} > \phi_{xy}^2, \quad \phi_{xx} > 0 \\ -1 & \phi_{xx}\,\phi_{yy} > \phi_{xy}^2, \quad \phi_{xx} < 0 \\ -j & \phi_{xx}\,\phi_{yy} < \phi_{xy}^2 \end{cases}$$

2. θ - *Ln r Transformation.* The requirements imposed by Eqns (B.5) on $\phi(x,y)$ are rather restrictive. One transformation which satisfies the conditions is the θ - ln r transformation where

$$u = \ln[\sqrt{x^2 + y^2}\,]\,, \qquad v = -\tan^{-1}\left[\frac{y}{x}\right] \tag{B.7}$$

The resulting phase filter function is

$$\phi(x, y) = \frac{2\pi}{\lambda f_L}\{ x \ln [\sqrt{x^2 + y^2}] - y \tan^{-1}(\frac{y}{x}) - x \} \qquad \text{(B.8)}$$

Figure B.2 presents this filter function using 2π contours.

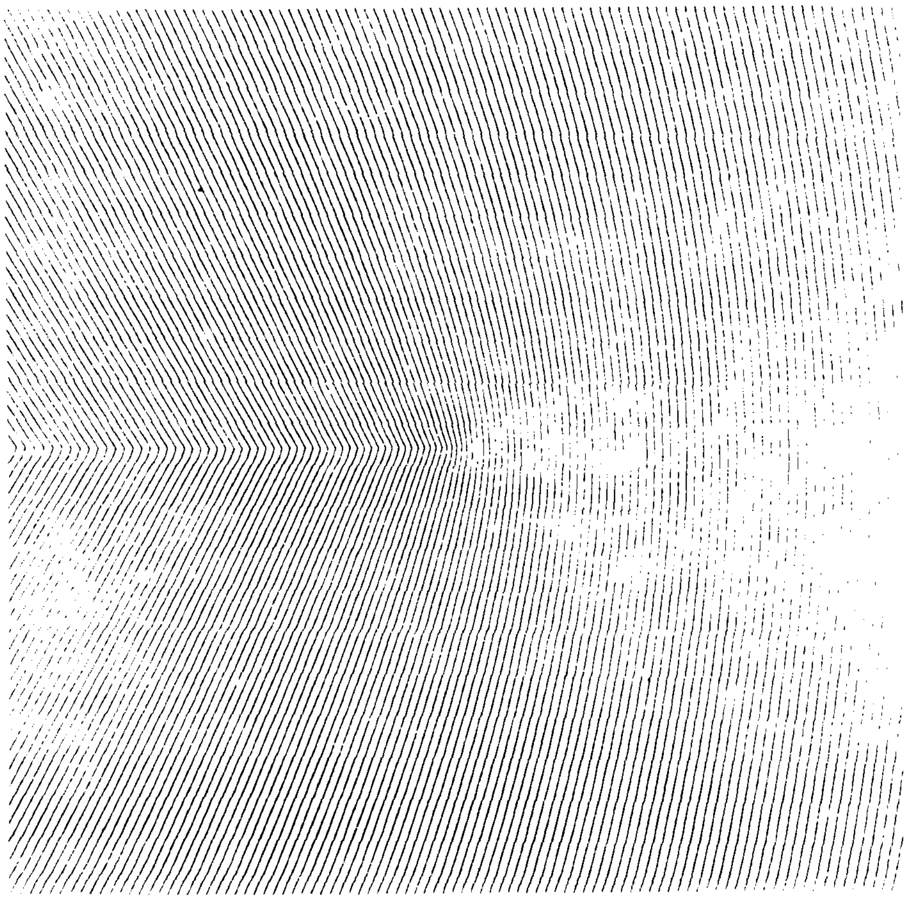


Figure B.2. θ - Ln r Coordinate Transformation Phase Filter

Phase filter for the [x, y] - to - [θ, Ln r] coordinate transformation. Plot generated from 2π contours of the phase function of Eqn (B.8).

3. *One-Dimensional Image Modification.* One-dimensional image transformation filters have no requirement such as Eqn (B.5). Optical implementation of the one-dimensional filters is identical to two-dimensional coordinate transformations. The exponential image modification filter is shown in Figure B.3 using 2π contour lines.

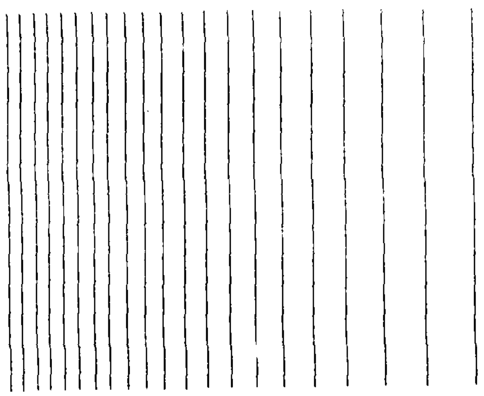


Figure B.3. Exponential Image Modification Phase Filter

One-dimensional phase filter for exponential expansion of input wavefront. Plot generated from 2π contours of the phase function.

Appendix C: Computer-Generated Interferograms

Information in this appendix was extracted from Lee [17:152-154] and Casasent *et al* [15].

1. *Definition.* The computer-generated interferogram is an optical technique for producing an arbitrary complex field using a binary amplitude transmittance. The complex field is created on a computer and an amplitude transmittance produced by computing the interference pattern of a reference wave and the complex field. The result is then output to a plotting device and photo-reduced onto an optical slide for use.

2. *Phase-Only Application.* Computer-generated interferograms are normally used to produce phase-only filters. In this application, the interference pattern of a desired phase function and an off-axis uniform amplitude plain reference wave is computed according to Eqn (C.1).

$$t(x,y) = 0.5 \{ 1 + \cos[2\pi \alpha x - \phi(x, y)] \} \qquad (C.1)$$

Maxima in this equation, which occur when

$$2\pi \alpha x - \phi(x,y) = 2\pi n , \qquad n = 0,1,2,...$$

form the binary amplitude transmittance. Output plots can then be photo-reduced onto the appropriate optical media. When this transmittance is illuminated with a planar wavefront, the desired combination of phase filter and input wavefront appear off-axis at an angle proportional to that of the reference wave α. Figure

C.1 displays a computer-generated interferogram computed on an AFIT VAX 11/785 computer using a *METALIB* plotting routine and output to an Imagen laser printer.

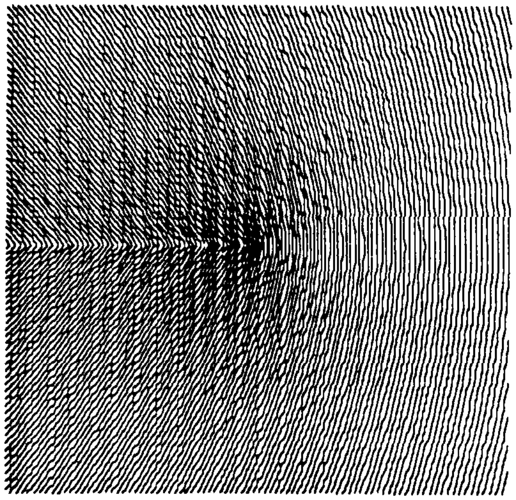


Figure C.1. θ - Ln r Computer-Generated Interferogram

Appendix D: Thesis Resources

This appendix lists the primary resources used to complete this thesis effort. The doppler data, computer resources and graphic resources were all made available by previous AFIT research sponsors.

A. *Doppler Data.* Doppler images were supplied by the AIr Force Wright Aeronautical Laboratory, Avionics Laboratory (AFWAL/ALTC).

B. *Computational Resources.* The bulk of the computer processing was performed on the AFIT Information Sciences Laboratory (ISL) VAX 11/780 computer. Contouring for generation of computer-generated interferograms was performed on an AFIT ICC Elxsi computer and AFIT Physics Department Sun II Workstation.

C. *Graphic Resources.* Edge and Hough transform images were displayed on the AFIT Department of Electrical and Computer Engineering Evans and Sutherland PS340 Graphics Workstation. Computer-generated interferograms were output on the AFIT Physics Department Imagen laser printer and photo-reduced in the photo-processing laboratory located in the AFIT Headquarters building.

Appendix E: VAX Programs

The Ada code developed during this thesis effort which specifically apply to the material presented in this document are listed in this appendix. Several packages used in some of the specifications but not list are edited versions of code developed in previous thesis work. They include the following:

Package Name	*Original Name*	*Original Author*
Data_Standard	Data_Standard	Tong [20]
Data_Conversions	Data_Conversions	Tong [20]
File_IO	File_IO	Tong [20]
SSI_IO	File_IO	Ruck [23]
Fourier2	Fourier_Transform_Handler	Kobel-Martin [22]

The Ada programs developed for this thesis are listed in the following pages of this appendix.

```
--**********************************************************************
--**********************************************************************
--
--                          PACKAGE HF1
--
--......................................................................
--
-- Purpose: Digital filters for straight line Hough images
--
--**********************************************************************

with DATA_STANDARD;     use DATA_STANDARD;
with float_math_lib;    use float_math_lib;
with text_io;           use text_io;
with integer_text_io;   use integer_text_io;

package HF1 is

procedure COMPARE( Input1: in  DATA_STANDARD.Image_Array_2d;
                   Input2: in  DATA_STANDARD.Image_Array_2d;
                   Match : out integer );

procedure DISTORT( Input : in     DATA_STANDARD.Image_Array_2d;
                   Rotate: in     integer;
                   Radius: in     integer;
                   Angle : in     integer;
                   Scale : in     float;
                   Output: in out DATA_STANDARD.Image_Array_2d );

procedure ESTIMATE( Input        : in  DATA_STANDARD.Image_Array_2d;
                    Template     : in  DATA_STANDARD.Image_Array_2d;
                    Rot_Min      : in integer;
                    Rot_Max      : in  integer;
                    Rot_Delta    : in  integer;
                    Rad_MIn      : in  integer;
                    Rad_Max      : in  integer;
                    Rad_Delta    : in  integer;
                    Ang_Min      : in  integer;
                    Ang_Max      : in  integer;
                    Ang_Delta    : in  integer;
                    Scale_Min    : in  float;
                    Scale_Max    : in  float;
                    Scale_Delta: in  float;
                    X_Location : out integer;
                    Y_Location : out integer;
                    Best_Rotate: out integer;
                    Best_Scale : out float;
                    Best_Match : out integer );

end HF1;

--<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< PACKGAE HF1 BODY >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
package body HF1 is

--*************************************************************************
--
--                              COMPARE
--
--.........................................................................
--
-- FUNCTION: Compare two iamges
--
-- INPUTS     (1) Input1 - First Image
--            (2) Input2 - Second Image
--
-- OUTPUT     (1) Match  - Percent match (integer 0 - 95)
--
--*************************************************************************

procedure COMPARE( Input1: in  DATA_STANDARD.Image_Array_2d;
                   Input2: in  DATA_STANDARD.Image_Array_2d;
                   Match : out integer ) is

Add, Sum                  : float := 0.0;
Count                     : integer := 1;
SIZE_ERROR                : exception;

begin -- COMPARE

-- [1] Check for size differences

if Input1'first(1) /= Input2'first(1) or Input1'last(1) /= Input2'last(1) then
   raise SIZE_ERROR;
elsif Input1'first(2) /= Input2'first(2) or Input1'last(2)/= Input2'last(2) then
   raise SIZE_ERROR;
end if;

-- [2] Compare images

for Theta in Input1'first(1)..Input1'last(1)
loop
   for Rho in Input1'first(2)..Input1'last(2)
   loop
      if Input1(Theta,Rho) > 0 then

         if Input2(Theta,Rho) > 0 then

            if Input1(Theta,Rho) < Input2(Theta,Rho) then
               Add := float( Input1(Theta,Rho) ) / float( Input2(Theta,Rho) );
            else
               Add := float( Input2(Theta,Rho) ) / float( Input2(Theta,Rho) );
            end if;

            Sum := Sum + Add;
```

```
         end if;

         Count := Count + 1;

      end if;
   end loop;
end loop;

if Count > 1 then
   Count := Count - 1;
end if;

Match := integer( 100.0 * Sum / float(Count) );

exception
   when SIZE_ERROR =>
      new_line;
      put_line(" *** SIZE ERROR ***");
      put_line(" Images of unequal sizes.");

end COMPARE;

--*******************************************************************************
--
--                                  DISTORT
--
--.............................................................................
--
-- FUNCTION: Adjust Hough space for a give shift in the image space
--
-- INPUTS:   (1) Input  - Input straight line Hough image
--           (2) Rotate - Rotation in image space (degrees)
--           (3) Radius - Shift radius in image space (pixels)
--           (4) Angle  - Shift angle in image space (degrees)
--           (5) Scale  - Scale in image space (fraction)
--
-- OUTPUTS:  (1) Output - Adjusted straight line Hough image
--
--*******************************************************************************

procedure DISTORT( Input : in     DATA_STANDARD.Image_Array_2d;
                   Rotate: in     integer;
                   Radius: in     integer;
                   Angle : in     integer;
                   Scale : in     float;
                   Output: in out DATA_STANDARD.Image_Array_2d ) is

Rad, Ang             : float;
Rho_In, Theta_In     : float;
Rho_Out, Theta_Out   : integer;

begin -- DISTORT
```

```
-- [1] Set Output to zero

for Theta in Output'first(1)..Output'last(1)
loop
   for Rho in Output'first(2)..Output'last(2)
   loop
      Output(Theta,Rho) := 0;
   end loop;
end loop;

-- [2] Distort image

Rad := float( Radius );
Ang := float( Angle  );

for Theta in Input'first(1)..Input'last(1)
loop
   for Rho in Input'first(2)..Input'last(2)
   loop
      if Input(Theta,Rho) > 0 then

         Rho_In   := float(Rho);
         Theta_In := float(Theta);

         Rho_Out  := integer( (Rho_In + Rad * cosd(Theta_In - Ang)) * Scale );

         if Rho_Out < 0 then
            Theta_Out := Theta - Rotate + 180;
            Rho_Out   := -Rho_Out;
         else
            Theta_Out := Theta - Rotate;
         end if;
         if Theta_Out > 359 then
            Theta_Out := Theta_Out - 360;
         elsif Theta_Out < 0 then
            Theta_Out := Theta_Out + 360;
         end if;
         if Input(Theta,Rho) > Output(Theta,Rho) then
            Output(Theta_Out,Rho_Out) := Input(Theta,Rho);
         end if;

      end if;
   end loop;
end loop;

end DISTORT;
```

```
--*************************************************************************
--
--                              ESTIMATE
--
--.........................................................................
--
-- FUNCTION: Estimate location and scale of template in input image
--
--*************************************************************************

procedure ESTIMATE( Input       : in  DATA_STANDARD.Image_Array_2d;
                    Template    : in  DATA_STANDARD.Image_Array_2d;
                    Rot_Min     : in  integer;
                    Rot_Max     : in  integer;
                    Rot_Delta   : in  integer;
                    Rad_MIn     : in  integer;
                    Rad_Max     : in  integer;
                    Rad_Delta   : in  integer;
                    Ang_Min     : in  integer;
                    Ang_Max     : in  integer;
                    Ang_Delta   : in  integer;
                    Scale_Min   : in  float;
                    Scale_Max   : in  float;
                    Scale_Delta: in  float;
                    X_Location  : out integer;
                    Y_Location  : out integer;
                    Best_Rotate: out integer;
                    Best_Scale  : out float;
                    Best_Match  : out integer ) is

Output   : DATA_STANDARD.Image_Array_2d( Input'first(1)..Input'last(1),
                                         Input'first(2)..Input'last(2) );
Done     : DATA_STANDARD.Bit_Array( 1..4 ) := ( true, true, true, true );
Rotate   : integer := Rot_Min;
Radius   : integer := Rad_Min;
Angle    : integer := Ang_Min;
Scale    : float   := Scale_Min;
Match    : integer := 0;
Best_Rad: integer := 0;
Best_Ang: integer := 0;
Best_Mat: integer := 0;

begin -- ESTIMATE

Best_Scale := Scale;

while Rotate <= Rot_Max
loop

   while Radius <= Rad_Max
   loop

      while Angle <= Ang_Max
```

```
      loop

         while Scale <= Scale_Max
         loop

            DISTORT( Template, Rotate, Radius, Angle, Scale, Output );
            COMPARE( Output, Input, Match );

            if Match > Best_Mat then
               Best_Mat    := Match;
               Best_Scale  := Scale;
               Best_Rad    := Radius;
               Best_Ang    := Angle;
               Best_Rotate:= Rotate;
            end if;

            Scale := Scale + Scale_Delta;

         end loop;
         Scale := Scale_Min;

         Angle := Angle + Ang_Delta;

      end loop;
      Angle := Ang_Min;

      Radius := Radius + Rad_Delta;

   end loop;
   Radius := Rad_Min;

   Rotate := Rotate + Rot_Delta;

end loop;

Best_Match := Best_Mat;
X_Location := integer( float(Best_Rad) * cosd( float(Best_Ang) ) );
Y_Location := integer( float(Best_Rad) * sind( float(Best_Ang) ) );

end ESTIMATE;

--**************************** END PACKAGE HF1 *********************************

end HF1;
```

```
--**************************************************************************
--
--                              ESTIMATOR
--
--........................................................................
--
-- FUNCTION: Estimate location and scale of template in input image
--
--**************************************************************************

with DATA_STANDARD;     use DATA_STANDARD;
with FILE_IO;           use FILE_IO;
with SSI_IO;            use SSI_IO;
with HF1;               use HF1;
with integer_text_io;   use integer_text_io;
with float_text_io;     use float_text_io;
with text_io;           use text_io;

procedure ESTIMATOR is

Input, Template                 : DATA_STANDARD.Image_Array_2d(0..359, 0.,181);
Input_Count, Template_Count     : natural;
Input_Name, Template_Name       : DATA_STANDARD.Line_Form;

Choice                          : character;

Rot_Min                         : integer := 0;
Rot_Max                         : integer := 0;
Rot_Delta                       : integer := 10;
Rad_Min                         : integer := 0;
Rad_Max                         : integer := 0;
Rad_Delta                       : integer := 5;
Ang_Min                         : integer := 0;
Ang_Max                         : integer := 0;
Ang_Delta                       : integer := 10;
Scale_Min                       : float   := 1.0;
Scale_Max                       : float   := 1.0;
Scale_Delta                     : float   := 0.1;
X_Location                      : integer := 1;
Y_Location                      : integer := 1;
Best_Rotate                     : integer := 0;
Best_Scale                      : float  := 1.0;
Best_Match                      : integer := 1;

begin -- ESTIMATOR

-- [1] Get input HS file

new_line;
put_line(" ENTER NAME OF INPUT HS SSI FILE");
GET_FILENAME( Input_Name, Input_Count);
READ_SSI_IMAGE( Input_Name, Input_Count, Input);
```

```
-- [2] Get Template

new_line;
put_line(" ENTER NAME OF TEMPLATE");
GET_FILENAME( Template_Name, Template_Count );
READ_SSI_IMAGE( Template_Name, Template_Count, Template );

-- [3] Get Range

GET_LOOP:
loop

   new_line(3);
   put_line(" *** HOUGH SPACE ESTIMATOR *** ");
   new_line;

   put("     Input Name       ==> "); put(Input_Name(1..Input_Count));
   new_line;
   put("     Template Name     ==> "); put(Template_Name(1..Template_Count));
   new_line(2);

   put(" [a] Min Rotation Value = "); put(Rot_Min,3,10); new_line;
   put(" [b] Max Rotation Value = "); put(Rot_Max,3,10); new_line;
   put(" [c] Rotation Delta     = "); put(Rot_Delta,3,10); new_line(2);

   put(" [d] Min Radius Value   = "); put(Rad_Min,3,10); new_line;
   put(" [e] Max Radius Value   = "); put(Rad_Max,3,10); new_line;
   put(" [f] Radius Delta       = "); put(Rad_Delta,3,10); new_line(2);

   put(" [g] Min Angle Value    = "); put(Ang_Min,3,10); new_line;
   put(" [h] Max Angle Value    = "); put(Ang_Max,3,10); new_line;
   put(" [i] Angle Delta        = "); put(Ang_Delta,3,10); new_line(2);

   put(" [j] Min Scale          = "); put(Scale_Min,1,2,0); new_line;
   put(" [k] Max Scale          = "); put(Scale_Max,1,2,0); new_line;
   put(" [l] Scale Delta        = "); put(Scale_Delta,1,2,0); new_line(2);

   put_line(" [z] Perform Estimation...");
   put(" ENTER CHOICE > "); get(Choice); new_line;

   case Choice is

   when 'a' | 'A' =>
      put(" Enter min rotation value > ");
      get(Rot_Min);

   when 'b' | 'B' =>
      put(" Enter max rotation value > ");
      get(Rot_Max);

   when 'c' | 'C' =>
      put(" Enter rotation delta > ");
      get(Rot_Delta);
```

```
    when 'd' | 'D' =>
       put(" Enter min radius value > ");
       get(Rad_Min);

    when 'e' | 'E' =>
       put(" Enter max radius value > ");
       get(Rad_Max);

    when 'f' | 'F' =>
       put(" Enter radius delta > ");
       get(Rad_Delta);

    when 'g' | 'G' =>
       put(" Enter min angle value > ");
       get(Ang_Min);

    when 'h' | 'H' =>
       put(" Enter max angle value > ");
       get(Ang_Max);

    when 'i' | 'I' =>
       put(" Enter angle delta > ");
       get(Ang_Delta);

    when 'j' | 'J' =>
       put(" Enter min scale value > ");
       get(Scale_Min);

    when 'k' | 'K' =>
       put(" Enter max scale value > ");
       get(Scale_Max);

    when 'l' | 'L' =>
       put(" Enter scale delta > ");
       get(Scale_Delta);

    when 'z' | 'Z' =>
       exit GET_LOOP;

    when others => null;

    end case;

 end loop GET_LOOP;

 -- [4] Perform estimation

 put_line(" %REM - Estimating...");

 ESTIMATE( Input,      Template,
           Rot_Min,    Rot_Max,     Rot_Delta,
           Rad_Min,    Rad_Max,     Rad_Delta,
```

```
        Ang_Min,     Ang_Max,     Ang_Delta,
        Scale_Min,   Scale_Max,   Scale_Delta,
        X_Location,  Y_Location,  Best_Rotate,  Best_Scale,   Best_Match );

-- [5] Print output

new_line(2);
put(" X        = "); put(X_Location, 3, 10); new_line;
put(" Y        = "); put(Y_Location, 3, 10); new_line;
put(" Rotation = "); put(Best_Rotate, 3, 10); new_line;
put(" Scale    = "); put(Best_Scale, 1, 2, 0); new_line;
put(" Match    = "); put(Best_Match, 3, 10); new_line;

new_line(2);
put_line(" *** END ESTIMATOR ***");
new_line(2);

end ESTIMATOR;
```

```
--***********************************************************************
--
--                         FOURIER_RADON
--
--.......................................................................
--
-- Purpose:  To generate a Radon Transform through a 2D Fourier Transform
-- followed by a inverse Fourier Transform along the radial direction.
--
--***********************************************************************

with DATA_STANDARD;       use DATA_STANDARD;
with DATA_CONVERSIONS;    use DATA_CONVERSIONS;
with FOURIER2;            use FOURIER2;
with SSI_IO;              use SSI_IO;
with FILE_IO;             use FILE_IO;
with text_io;             use text_io;
with integer_text_io;     use integer_text_io;
with float_math_lib;      use float_math_lib;

procedure FOURIER_RADON is

Input, FT_Output          : DATA_STANDARD.Image_Array_2d( 1..256, 1..256 );
Output, FT_RT_Output      : DATA_STANDARD.Image_Array_2d( 0..359, 0..181 );
In_Work                   : DATA_STANDARD.Complex_Array_R2d( 1..256, 1..256 );
Out_Work                  : DATA_STANDARD.Complex_Array_R2d( 0..359, 0..181 );
Rho_Work                  : DATA_STANDARD.Complex_Array_R1d( 0..255 );
Output_Histogram          : DATA_STANDARD.Image_Array_1d( 0..255 );
Output_Histogram_float    : DATA_STANDARD.Array_1d( 0..255 );
Name                      : DATA_STANDARD.Line_Form;
Direction                 : FOURIER2.Transform_Type;
X_Center, Y_Center,
Nearest_X, Nearest_Y      : integer;
Max_Output                : integer := -10000;
Min_Output                : integer :=  10000;
Temp                      : float   :=  0.0;
Temp_Out,
X_Arg, Y_Arg,
X_Arg_Cos, X_Arg_Sin,
Y_Arg_Cos, Y_Arg_Sin,
In_Real, In_Img,
Real_Shift, Img_Shift     : float;
Char_Count                : natural;
Answer                    : character;
Edge_Value                : constant integer := 255;

begin -- FOURIER_RADON

-- [1] Get input image

new_line;
put_line(" ENTER SSI INPUT IMAGE");
GET_FILENAME( Name, Char_Count );
```

```
READ_SSI_IMAGE( Name, Char_Count, Input );

-- [2] Convert input to complex image array

for X in Input'first(1)..Input'last(1)
loop
   for Y in Input'first(2)..Input'last(2)
   loop
      if Input(X,Y) = Edge_Value then
         In_Work(X,Y).Real_N := 10.0;
    else
         In_Work(X,Y).Real_N := 0.0;
      end if;
         In_Work(X,Y).Img_N := 0.0;
   end loop;
end loop;

-- [3] Take Fourier Transform

put_line(" %REM - Taking 2D Fourier Transform... ");

X_Arg := 4.0 * atan(1.0);
Y_Arg := 4.0 * atan(1.0);

for X in In_Work'first(1)..In_Work'last(1)
loop
   for Y in In_Work'first(2)..In_Work'last(2)
   loop

      X_Arg_Cos := cos( float(X) * X_Arg );
      X_Arg_Sin := -sin( float(X) * X_Arg );
      Y_Arg_Cos := cos( float(Y) * Y_Arg );
      Y_Arg_Sin := -sin( float(Y) * Y_Arg );

      Real_Shift := X_Arg_Cos * Y_Arg_Cos - X_Arg_Sin * Y_Arg_Sin;
      Img_Shift  := X_Arg_Sin * Y_Arg_Sin + X_Arg_Cos * Y_Arg_Cos;

      In_Real := In_Work(X,Y).Real_N;
      In_Img  := In_Work(X,Y).Img_N;

      In_Work(X,Y).Real_N := In_Real * Real_Shift - In_Img * Img_Shift;
      In_Work(X,Y).Img_N  := In_Img * Real_Shift + In_Real * Img_Shift;

   end loop;
end loop;

Direction := forward;

Two_DFT( In_Work, Direction);

X_Arg := 4.0 * atan(1.0);
Y_Arg := 4.0 * atan(1.0);
```

```
for X in In_Work'first(1)..In_Work'last(1)
loop
   for Y in In_Work'first(2)..In_Work'last(2)
   loop

      X_Arg_Cos := cos( float(X) * X_Arg );
      X_Arg_Sin := sin( float(X) * X_Arg );
      Y_Arg_Cos := cos( float(Y) * Y_Arg );
      Y_Arg_Sin := sin( float(Y) * Y_Arg );

      Real_Shift := X_Arg_Cos * Y_Arg_Cos - X_Arg_Sin * Y_Arg_Sin;
      Img_Shift  := X_Arg_Sin * Y_Arg_Sin + X_Arg_Cos * Y_Arg_Cos;

      In_Real := In_Work(X,Y).Real_N;
      In_Img  := In_Work(X,Y).Img_N;

      In_Work(X,Y).Real_N := In_Real * Real_Shift - In_Img * Img_Shift;
      In_Work(X,Y).Img_N  := In_Img * Real_Shift + In_Real * Img_Shift;

   end loop;
end loop;


new_line;
put(" SAVE 2D FOURIER TRANSFORM? (y/n) > "); get(Answer);

case Answer is
when 'y' | 'Y' =>

   for X in FT_Output'first(1)..FT_Output'last(1)
   loop
      for Y in FT_Output'first(2)..FT_Output'last(2)
      loop

         Temp := float( In_Work(X,Y).Real_N**2 + In_Work(X,Y).Img_N**2 );
         FT_Output(X,Y) := integer( sqrt(Temp) + 0.5 );

         if FT_Output(X,Y) > Max_Output then
            Max_Output := FT_Output(X,Y);
         end if;

         if FT_Output(X,Y) < Min_Output then
            Min_Output := FT_Output(X,Y);
         end if;

      end loop;
   end loop;

   for X in FT_Output'first(1)..FT_Output'last(1)
   loop
      for Y in FT_Output'first(2)..FT_Output'last(2)
      loop
```

```
            Temp := 255.0 * (float(FT_Output(X,Y)-Min_Output)) / float(Max_Output);
            FT_Output(X,Y) := integer( Temp );
         end loop;
      end loop;

      Max_Output := -10000;
      Min_Output :=  10000;

      new_line;
      put_line(" ENTER NAME OF FT FILE");
      GET_FILENAME( Name, Char_Count );
      SAVE_SSI_IMAGE( Name, Char_Count, FT_Output);

   when others =>

      null;

   end case;

   -- [4] Convert to Polar Coordinates

   put_line(" %REM - Converting to polar coordinates...");

   --RECT_TO_POLAR( In_Work, Out_Work );

   X_Center := ( In_Work'last(1) - In_Work'first(1) ) / 2;
   Y_Center := ( In_Work'last(2) - In_Work'first(2) ) / 2;

   for Theta in Out_Work'first(1)..Out_Work'last(1)
   loop
      for Rho in Out_Work'first(2)..Out_Work'last(2)
      loop

         Out_Work(Theta,Rho).Real_N := 0.0;
         Out_Work(Theta,Rho).Img_N  := 0.0;

         Nearest_X := integer( float(Rho) * cosd(float(Theta)) + 0.5 ) + X_Center;
         Nearest_Y := integer( float(Rho) * sind(float(Theta)) + 0.5 ) + Y_Center;

         if Nearest_X >= In_Work'first(1) and Nearest_X <= In_Work'last(1) and
            Nearest_Y >= In_Work'first(2) and Nearest_Y <= In_Work'last(2) then

            Out_Work(Theta,Rho).Real_N := In_Work(Nearest_X,257-Nearest_Y).Real_N;
            Out_Work(Theta,Rho).Img_N  := In_Work(Nearest_X,257-Nearest_Y).Img_N;

         end if;

      end loop;
   end loop;

   new_line;
   put(" SAVE POLAR 2D FOURIER TRANSFORM? (y/n) > "); get(Answer);
```

```
case Answer is
when 'y' | 'Y' =>

   for Theta in FT_RT_Output'first(1)..FT_RT_Output'last(1)
   loop
      for Rho in FT_RT_Output'first(2)..FT_RT_Output'last(2)
      loop

         Temp := float( Out_Work(Theta,Rho).Real_N**2
                      + Out_Work(Theta,Rho).Img_N**2 );
         FT_RT_Output(Theta,Rho) := integer( sqrt(Temp) + 0.5 );

         if FT_RT_Output(Theta,Rho) > Max_Output then
            Max_Output := FT_RT_Output(Theta,Rho);
         end if;

         if FT_RT_Output(Theta,Rho) < Min_Output then
            Min_Output := FT_RT_Output(Theta,Rho);
         end if;

      end loop;
   end loop;

   for Theta in FT_RT_Output'first(1)..FT_RT_Output'last(1)
   loop
      for Rho in FT_RT_Output'first(2)..FT_RT_Output'last(2)
      loop
         Temp := 255.0 * ( float( FT_RT_Output(Theta,Rho) - Min_Output ) )
                 / float(Max_Output);
         FT_RT_Output(Theta,Rho) := integer( Temp );
      end loop;
   end loop;

   Max_Output := -10000;
   Min_Output :=  10000;

   new_line;
   put_line(" ENTER NAME OF POLAR FT FILE");
   GET_FILENAME( Name, Char_Count );
   SAVE_SSI_IMAGE( Name, Char_Count, FT_RT_Output);

when others =>

   null;

end case;

-- [5] Take inverse Fourier Transform along the Rho direction

put_line(" %REM - Taking radial inverse Fourier Transform...");

for Rho in Rho_Work'first..Rho_Work'last
loop
```

```
      Rho_Work(Rho).Real_N := 0.0;
      Rho_Work(Rho).Img_N  := 0.0;
   end loop;

   Direction := inverse;

   for Theta in Out_Work'first(1)..Out_Work'last(1)
   loop

--    for Rho in Out_Work'first(2)..Out_Work'last(2)
      for Rho in 0..127
      loop
         Rho_Work(Rho).Real_N := Out_Work(Theta,Rho+1).Real_N;
         Rho_Work(Rho).Img_N  := Out_Work(Theta,Rho+1).Img_N;
      end loop;

      One_DFT( Rho_Work, Direction );

      for Rho in Output'first(2)..Output'last(2)
--    for Rho in 0..127
      loop

--          Temp_Out := Rho_Work(255-Rho).Real_N**2 + Rho_Work(255-Rho).Img_N**2;
         Temp_Out := Rho_Work(Rho).Real_N**2 + Rho_Work(Rho).Img_N**2;
         Output(Theta,Rho) := integer( sqrt(Temp_Out) + 0.5 );

         if Max_Output < Output(Theta,Rho) then
            Max_Output := Output(Theta,Rho);
         end if;
         if Min_Output > Output(Theta,Rho) then
            Min_Output := Output(Theta,Rho);
         end if;

      end loop;

--    for Rho in 129..181
--    loop
--       Output(Theta,Rho) := 0;
--    end loop;

   end loop;

   new_line;
   put("Max Output = "); put(Max_Output,3,10); new_line;
   put("Min Output = "); put(Min_Output,3,10);

   -- [6] Save in 2D Radon file

   for Theta in Output'first(1)..Output'last(1)
   loop
      for Rho in Output'first(2)..Output'last(2)
      loop
         Temp := float( Output(Theta,Rho) - Min_Output ) * 255.0;
```

```
      Temp := Temp / float( Max_Output - Min_Output );
      Output(Theta,Rho) := integer( Temp );
   end loop;
end loop;

-- [6.1] form histogram

new_line;
put(" FORM HISTOGRAM? (y/n) > "); get(Answer); new_line;

case Answer is
when 'y' | 'Y' =>

   HISTO_GRAM1_2D( Output, Output_Histogram );

   for Index in Output_Histogram'first..Output_Histogram'last
   loop
      Output_Histogram_float(Index) := float( Output_Histogram(Index) );
   end loop;

   new_line;
   put_line(" ENTER NAME OF MATRIXX HISTOGRAM FILE");
   GET_FILENAME( Name, Char_Count );
   MATRIX_X_PLOT1D( Name, Char_Count, Output_Histogram_float );

when others => null;

end case;

--

new_line(2);
put_line(" ENTER RADON SSI FILENAME");
GET_FILENAME( Name, Char_Count );
SAVE_SSI_IMAGE( Name, Char_Count, Output );

new_line(2):
put_line(" *** END FOURIER_RADON ***");

end FOURIER_RADON;
```

```
--***********************************************************************
--
--                              INV_FR
--
--.......................................................................
--
-- Purpose:  To generate the orginal image from the Radon Transform through
-- a radial 1D Fourier Transform followed by a polar to reactangular coordinate
-- transformation and an inverse 2D Fourier Transform.
--
--***********************************************************************

with DATA_STANDARD;      use DATA_STANDARD;
with DATA_CONVERISONS;   use DATA_CONVERSIONS;
with FOURIER2;           use FOURIER2;
with SSI_IO;             use SSI_IO;
with FILE_IO;            use FILE_IO;
with text_io;            use text_io;
with integer_text_io;    use integer_text_io;
with float_math_lib;     use float_math_lib;

procedure INV_FR is

Input, RFT_Output        : DATA_STANDARD.Image_Array_2d( 0..359, 0..181 );
Output, FT_Output        : DATA_STANDARD.Image_Array_2d( 1..256, 1..256 );
In_Work                  : DATA_STANDARD.Complex_Array_R2d( 0..359, 0..127 );
Out_Work                 : DATA_STANDARD.Complex_Array_R2d( 1..256, 1..256 );
Rho_Work                 : DATA_STANDARD.Complex_Array_R1d( 0..255 );
Output_Histogram         : DATA_STANDARD.Image_Array_1d( 0..255 );
Output_Histogram_float   : DATA_STANDARD.Array_1d( 0..255 );
Name                     : DATA_STANDARD.Line_Form;
Direction                : FOURIER2.Transform_Type;

Nearest_Rho,
Nearest_Theta            : integer;
Block_DC                 : constant integer := 2;
X_Center, Y_Center       : constant integer := 128;
Max_Output               : integer := -10000;
Min_Output               : integer :=  10000;
Temp                     : float   :=  0.0;
Temp_Out,
pi                       : constant float := 4.0 * atan(1.0);
X_Arg_Cos, X_Arg_Sin.
Y_Arg_Cos, Y_Arg_Sin,
In_Real, In_Img,
Real_Shift, Img_Shift    : float;
X1, Y1                   : float;
Char_Count               : natural;
Answer                   : character;
Edge_Value               : constant integer := 255;

begin -- INV_FR
```

```
-- [1] Get input image

new_line;
put_line(" ENTER RADON INPUT IMAGE");
GET_FILENAME( Name, Char_Count );
READ_SSI_IMAGE( Name, Char_Count, Input );

-- [2] Convert input to complex image array and take Radial Fourier Transform

put_line(" %REM - Taking radial Fourier Transform...");

Direction := forward;

for Theta in In_Work'first(1)..In_Work'last(1)
loop

   for Rho in In_Work'first(2)..In_Work'last(2)
   loop

--       Y_Arg_Cos := cos( float(Rho) * pi );
--       Y_Arg_Sin := -sin( float(Rho) * pi );

      Y_Arg_Cos := 1.0;
      Y_Arg_SIn := 1.0;

      Rho_Work(Rho).Real_N := float(Input(Theta,Rho)) * Y_Arg_Cos;
      Rho_Work(Rho).Img_N  := float(Input(Theta,Rho)) * Y_Arg_Sin;

   end loop;

   for Rho in In_Work'last(2)+1..Rho_Work'last
   loop
      Rho_Work(Rho).Real_N := 0.0;
      Rho_Work(Rho).Img_N  := 0.0;
   end loop;

   One_DFT( Rho_Work, Direction );

   for Rho in In_Work'first(2)..In_Work'last(2)
   loop
         In_Work(Theta,Rho).Real_N := Rho_Work(Rho).Real_N;
         In_Work(Theta,Rho).Img_N  := Rho_Work(Rho).Img_N;
   end loop;

end loop;

new_line;
put(" SAVE RADIAL FOURIER TRANSFORM? (y/n) > "); get(Answer);

case Answer is
when 'y' | 'Y' =>
```

```
for Theta in RFT_Output'first(1)..RFT_Output'last(1)
loop
   for Rho in RFT_Output'first(2)..In_Work'last(2)
   loop

      Temp := float( In_Work(Theta,Rho).Real_N**2
               + In_Work(Theta,Rho).Img_N**2 );
      RFT_Output(Theta,Rho) := integer( sqrt(Temp) + 0.5 );

      if RFT_Output(Theta,Rho) > Max_Output then
         Max_Output := RFT_Output(Theta,Rho);
      end if;

      if RFT_Output(Theta,Rho) < Min_Output then
         Min_Output := RFT_Output(Theta,Rho);
      end if;

   end loop;

   for Rho in In_Work'last(2) + 1 .. RFT_Output'last(2)
   loop
      RFT_Output(Theta,Rho) := Min_Output;
   end loop;

end loop;

for Theta in RFT_Output'first(1)..RFT_Output'last(1)
loop
   for Rho in RFT_Output'first(2)..RFT_Output'last(2)
   loop
      Temp := 255.0 * ( float( RFT_Output(Theta,Rho)
               - Min_Output ) ) / float(Max_Output);
      RFT_Output(Theta,Rho) := integer( Temp );
   end loop;
end loop;

Max_Output := -10000;
Min_Output :=  10000;

new_line;
put_line(" ENTER NAME OF RFT FILE");
GET_FILENAME( Name, Char_Count );
SAVE_SSI_IMAGE( Name, Char_Count, RFT_Output);

when others =>

   null;

end case;

-- [3] Polar to Cartesian
```

```
for X in Out_Work'first(1)..Out_Work'last(1)
loop
   for Y in Out_Work'first(2)..Out_Work'last(2)
   loop

      X1 := float(X - X_Center);
      Y1 := float(257 - Y - Y_Center);
      Nearest_Rho := integer( sqrt( X1**2 + Y1**2 ) );

      if X1 = 0.0 and Y1 >= 0.0 then
         Nearest_Theta := 90;
      elsif X1 = 0.0 and Y1 < 0.0 then
         Nearest_Theta := -90;
      else
         Nearest_Theta := integer( atan2d( Y1, X1 ) );
      end if;

      if Nearest_Theta < 0 then
         Nearest_Theta := Nearest_Theta + 360;
      end if;

      if Nearest_Theta in In_Work'range(1) and Nearest_Rho in In_Work'range(2)
      then
         Out_Work(X,Y).Real_N := In_Work(Nearest_Theta,Nearest_Rho).Real_N;
         Out_Work(X,Y).Img_N  := In_Work(Nearest_Theta,Nearest_Rho).Img_N;
      end if;

   end loop;
end loop;

new_line;
put(" SAVE RECT FOURIER TRANSFORM? (y/n) > "); get(Answer);

case Answer is
when 'y' | 'Y' =>

   for X in FT_Output'first(1)..FT_Output'last(1)
   loop
      for Y in FT_Output'first(2)..FT_Output'last(2)
      loop

         Temp := float( Out_Work(X,Y).Real_N**2 + Out_Work(X,Y).Img_N**2 );
         FT_Output(X,Y) := integer( sqrt(Temp) + 0.5 );

         if FT_Output(X,Y) > Max_Output then
            Max_Output := FT_Output(X,Y);
         end if;

         if FT_Output(X,Y) < Min_Output then
            Min_Output := FT_Output(X,Y);
         end if;
```

```
      end loop;
   end loop;

   for X in FT_Output'first(1)..FT_Output'last(1)
   loop
      for Y in FT_Output'first(2)..FT_Output'last(2)
      loop
         Temp := 255.0 * ( float( FT_Output(X,Y) - Min_Output ) )
                 / float(Max_Output);
         FT_Output(X,Y) := integer( Temp );
      end loop;
   end loop;

   Max_Output := -10000;
   Min_Output :=  10000;

   new_line;
   put_line(" ENTER NAME OF FT FILE");
   GET_FILENAME( Name, Char_Count );
   SAVE_SSI_IMAGE( Name, Char_Count, FT_Output);

when others =>

   null;

end case;

-- [4] Take 2D inverse Fourier Transform

put_line(" %REM - Taking 2D Fourier Transform... ");

for X in Out_Work'first(1)..Out_Work'last(1)
loop
   for Y in Out_Work'first(2)..Out_Work'last(2)
   loop

      X_Arg_Cos := cos( float(X) * pi );
      X_Arg_Sin := -sin( float(X) * pi );
      Y_Arg_Cos := cos( float(Y) * pi );
      Y_Arg_Sin := -sin( float(Y) * pi );

      Real_Shift := X_Arg_Cos * Y_Arg_Cos - X_Arg_Sin * Y_Arg_Sin;
      Img_Shift  := X_Arg_Sin * Y_Arg_Sin + X_Arg_Cos * Y_Arg_Cos;

      In_Real := Out_Work(X,Y).Real_N;
      In_Img  := Out_Work(X,Y).Img_N;

      Out_Work(X,Y).Real_N := In_Real * Real_Shift - In_Img * Img_Shift;
      Out_Work(X,Y).Img_N  := In_Img * Real_Shift + In_Real * Img_Shift;

   end loop;
end loop;
```

```
Direction := inverse;

Two_DFT( Out_Work, Direction);

-- [4.1] Block DC component

for X in X_Center - Block_DC .. X_Center + Block_DC
loop
   for Y in Y_Center - Block_DC .. Y_Center + Block_DC
   loop
      Out_Work(X,Y).Real_N := 0.0;
      Out_Work(X,Y).Img_N  := 0.0;
   end loop;
end loop;

for X in Output'first(1)..Output'last(1)
loop
   for Y in Output'first(2)..Output'last(2)
   loop

      Temp := float( Out_Work(X,Y).Real_N**2 + Out_Work(X,Y).Img_N**2 );
      Output(X,Y) := integer( sqrt(Temp) + 0.5 );

      if Output(X,Y) > Max_Output then
         Max_Output := Output(X,Y);
      end if;

      if Output(X,Y) < Min_Output then
         Min_Output := Output(X,Y);
      end if;

   end loop;
end loop;

for X in Output'first(1)..Output'last(1)
loop
   for Y in Output'first(2)..Output'last(2)
   loop
      Temp := 255.0 * ( float( Output(X,Y) - Min_Output ) ) / float(Max_Output);
      Output(X,Y) := integer( Temp );
   end loop;
end loop;

-- [5] form histogram

new_line;
put(" FORM HISTOGRAM? (y/n) > "); get(Answer); new_line;

case Answer is
when 'y' | 'Y' =>

   HISTO_GRAM1_2D( Output, Output_Histogram );
```

```
   for Index in Output_Histogram'first..Output_Histogram'last
   loop
      Output_Histogram_float(Index) := float( Output_Histogram(Index) );
   end loop;

   new_line;
   put_line(" ENTER NAME OF MATRIXX HISTOGRAM FILE");
   GET_FILENAME( Name, Char_Count );
   MATRIX_X_PLOT1D( Name, Char_Count, Output_Histogram_float );

when others => null;

end case;

-- [6] Save image

new_line(2);
put_line(" ENTER SSI FILENAME");
GET_FILENAME( Name, Char_Count );
SAVE_SSI_IMAGE( Name, Char_Count, Output );

new_line(2);
put_line(" *** END INV_FR ***");

end INV_FR;
```

```
--***************************************************************************
--
--                                  RADON2
--
--.........................................................................
--
--  Purpose: Perform 2D Radon Transform on input image
--
--***************************************************************************

with DATA_STANDARD;       use DATA_STANDARD;
with FILE_IO;             use FILE_IO;
with SSI_IO;              use SSI_IO;
with float_math_lib;      use float_math_lib;
with text_io;             use text_io;
with integer_text_io;     use integer_text_io;

procedure RADON2 is

Input                     : DATA_STANDARD.Image_Array_2d( 1..256, 1..256 );
Output                    : DATA_STANDARD.Image_Array_2d( 0..359, 0..181 );
Theta_Float, X, Y         : float;
Rho_Out, Temp_Out         : integer;
Max_Output                : integer := 0;
Min_Output                : integer := 2000;
Edge_Value                : integer := 255;
Shift                     : integer := 128;
Name                      : DATA_STANDARD.Line_Form;
Char_Count                : natural;

begin -- RADON2

-- [1] Get input image

new_line;
put_line(" NAME OF SSI IMAGE ?");
GET_FILENAME(Name, Char_Count);
READ_SSI_IMAGE(Name, Char_Count, Input);

-- [2] Set Radon space to zero

new_line;
put_line(" %REM - Computing Radon Transform...");

for Theta in Output'first(1)..Output'last(1)
loop
   for Rho in Output'first(2)..Output'last(2)
   loop
      Output(Theta, Rho) := 0;
   end loop;
end loop;

-- [3] Take Radon Transform
```

```
new_line;

for X1 in Input'first(1)..Input'last(1)
loop
   for Y1 in Input'first(2)..Input'last(2)
   loop
      if Input(X1,Y1) > 0 then

         for Theta in Output'first(1)..Output'last(1)
         loop

            X           := float(X1 - Shift);
            Y           := float(257 - Y1 - Shift);
            Theta_Float := float(Theta);
            Rho_Out := integer( X * cosd(Theta_Float) + Y * sind(Theta_Float) );

            if Rho_Out > 0 then
               Output(Theta,Rho_Out) := Output(Theta,Rho_Out) + Input(X1,Y1);
            elsif Rho_Out = 0 and Theta < 180 then
               Output(Theta,Rho_Out) := Output(Theta,Rho_Out) + Input(X1,Y1);
            end if;

         end loop;
      end if;
   end loop;
end loop;

new_line;

for Theta in Output'first(1)..Output'last(1)
loop
   for Rho in Output'first(2)..Output'last(2)
   loop

      if Max_Output < Output(Theta,Rho) then
         Max_Output := Output(Theta,Rho);
      end if;

      if Min_Output > Output(Theta,Rho) then
         Min_Output := Output(Theta,Rho);
      end if;

   end loop;
end loop;

new_line;

for Theta in Output'first(1)..Output'last(1)
loop
   for Rho in Output'first(2)..Output'last(2)
   loop
      Output(Theta,Rho)
```

```
          := integer( 255.0 * float(Output(Theta,Rho)) / float(Max_Output) );
   end loop;
end loop;

put("MINIMUM VALUE = "); put(Min_Output); new_line;
put("MAXIMUM VALUE = "); put(Max_Output); new_line(2);

-- [4] Save Radon image

new_line;
put_line(" OUTPUT FILENAME?");
GET_FILENAME(Name, Char_Count);
SAVE_SSI_IMAGE(Name, Char_Count, Output);
new_line;

put_line(" *** END 2D RADON TRANSFORM ***");
new_line;

end RADON2;
```

```
--*************************************************************************
--
--                    THRESHOLD_HS IMAGE (SSI - Format)
--
--.........................................................................
--
--  Purpose:  Threshold Hough Transform Image
--
--*************************************************************************

with DATA_STANDARD;      use DATA_STANDARD;
with FILE_IO;            use FILE_IO;
with SSI_IO;             use SSI_IO;
with text_io;            use text_io;
with integer_text_io;    use integer_text_io;

procedure THRESHOLD_HS is

Input                    : DATA_STANDARD.Image_Array_2d(0..359,0..181);
Name                     : DATA_STANDARD.Line_Form;
Lower, Upper, Max        : integer range 0..255;
Char_Count               : natural;
Answer                   : character;

begin -- THRESHOLD_HS

-- [1] Get image

new_line;
put_line(" NAME OF THE INPUT SSI FILE?");
GET_FILENAME(Name, Char_Count);
READ_SSI_IMAGE(Name, Char_Count, Input);

for Theta in Input'first(1)..Input'last(1)
loop
   for Rho in Input'first(2)..Input'last(2)
   loop
      if Max < Input(Theta, Rho) then
          Max := Input(Theta, Rho);
      end if;
   end loop;
end loop;

new_line;
put("MAXIMUM VALUE = "); put(Max); new_line;

-- [2] Get threshold value

begin

GET_LOWER:
loop
   new_line;
```

```
   put_line(" ENTER LOWER BOUND (0 - 255)");
   get(Lower);
   put(" LOWER BOUND = ");
   put(Lower);
   put("? (Y/N) => ");
   get(Answer);

   case Answer is
      when 'Y' | 'y' =>
         exit GET_LOWER;
      when others =>
         null;
   end case;

end loop GET_LOWER;

GET_UPPER:
loop
   new_line;
   put_line(" ENTER UPPER BOUND (0 - 255)");
   get(Upper);
   put(" UPPER BOUND = ");
   put(Upper);
   put("? (Y/N) => ");
   get(Answer);

   case Answer is
      when 'Y' | 'y' =>
         exit GET_UPPER;
      when others =>
         null;
   end case;

end loop GET_UPPER;

exception
   when Data_Error =>
      put_line(" Invalid entry.");
      put_line(" Enter positive integer value (0 - 255).");
end;

-- [3] Threshold and save in SSI format

put_line("Begin thresholding...");

for Rho in Input'first(2)..Input'last(2)
loop
   for Theta in Input'first(1)..Input'last(1)
   loop
      if Input(Theta,Rho) > Upper or Input(Theta,Rho) < Lower then
         Input(Theta,Rho) := 0;
      end if;
   end loop;
```

```
end loop;

new_line;
put_line(" NAME OF OUTPUT SSI FILE?");
GET_FILENAME(Name, Char_Count);
SAVE_SSI_IMAGE(Name, Char_Count, Input);

new_line;
put_line(" *** END THRESHOLD_HS ***");
new_line;

end THRESHOLD_HS;
```

```
--***********************************************************************
--
--                              OCCLUDE
--
--.......................................................................
--
-- Purpose: To block a section of input image
--
--***********************************************************************

with DATA_STANDARD;     use DATA_STANDARD;
with FILE_IO;           use FILE_IO;
with SSI_IO;            use SSI_IO;
with text_io;           use text_io;
with integer_text_io;   use integer_text_io;
with float_math_lib;    use float_math_lib;

procedure OCCLUDE is

Input                       : DATA_STANDARD.Image_Array_2d( 1..256, 1..256 );
Name                        : DATA_STANDARD.Line_Form;
Char_Count                  : natural;
Lower_Left_X, Lower_Left_Y  : integer;
Box_Size                    : integer;
X_Center, Y_Center          : integer;
X_Float, Y_Float            : float;
Radius                      : integer;
Answer                      : character;
Choice                      : integer range 0..3;

begin -- OCCLUDE

-- [1] Get input image

new_line;
put_line(" ENTER NAME OF SSI INPUT");
GET_FILENAME( Name, Char_Count );
READ_SSI_IMAGE( Name, Char_Count, Input );

-- [2] Get blocking parameters

GET_LOOP:
loop

   new_line(3);
   put_line(" *** OCCLUSION TYPES ***");
   new_line;
   put("    [1] Circular"); new_line(2);
   put("    [2] Square"); new_line(2);
   put(" ENTER CHOICE > "); get(Choice); new_line;

   case Choice is
   when 1 | 2  => exit GET_LOOP;
```

```
      when others => null;
      end case;

   end loop GET_LOOP;

   case Choice is
   when 1 =>

      CIRC:
      loop

         put_line(" *** Circle Parameters ***");
         put("  [1] Center = ("); put(X_Center,4,10);
         put(","); put(Y_Center,4,10); put(")"); new_line(2);
         put("  [2] Circle Radius = "); put(Radius,3,10); new_line(2);
         put("  [0] Continue"); new_line(2);
         put(" Enter Choice => "); get(Choice); new_line;

         case Choice is
         when 1 =>

            CENTER_X:
            loop

               new_line;
               put_line(" ENTER X Value (-127..128) => "); get(X_Center);
               put(" X value = "); put(X_Center,4,10); put(" (Y/N)?"); get(Answer);

               case Answer is
               when 'y' | 'Y' => exit CENTER_X;
               when others    => null;
               end case;

            end loop CENTER_X;

            CENTER_Y:
            loop

               new_line;
               put_line(" ENTER Y Value (-127..128) => "); get(Y_Center);
               put(" Y value = "); put(Y_Center,4,10); put(" (Y/N)?"); get(Answer);

               case Answer is
               when 'y' | 'Y' => exit CENTER_Y;
               when others    => null;
               end case;

            end loop CENTER_Y;

         when 2 =>

            GET_RADIUS:
            loop
```

```
            new_line;
            put_line(" Enter Radius Value (1..128) => "); get(Radius);
            put(" Radius = "); put(Radius,3,10); put(" (Y/N)?"); get(Answer);

            case Answer is
            when 'y' | 'Y' => exit GET_RADIUS;
            when others    => null;
            end case;

         end loop GET_RADIUS;

      when others => null;

      end case;

   end loop CIRC;

   put_line(" %REM - Setting values to zero...");

   for X in Input'first(1)..Input'last(1)
   loop
      for Y in Input'first(2)..Input'last(2)
      loop

         X_Float := ( float(X) - 128.0 ) - float(X_Center);
         Y_Float := ( 257.0 - float(Y) - 128.0 ) - float(Y_Center);

         if sqrt( X_Float**2 + Y_Float**2 ) <= float(Radius) then
            Input(X,Y) := 0;
         end if;

      end loop;
   end loop;

when 2 =>

   BOX:
   loop

      put_line(" *** Box Parameters ***");
      put("  [1] Lower Left Corner = ("); put(Lower_Left_X,4,10);
      put(","); put(Lower_Left_Y,4,10); put(")");
      put(")"); new_line(2);
      put("  [2] Size = "); put(Box_Size,3,10); new_line(2);
      put("  [0] Continue"); new_line(2);
      put(" Enter Choice => "); get(Choice); new_line;

      case Choice is
      when 1 =>

         LOWER_LEFT:
         loop
```

```
            new_line;
            put_line(" Enter X value of lower left corner (-127..128) => ");
            get(Lower_Left_X);

            put_line(" Enter Y value lower left corner (-127..128) => ");
            get(Lower_Left_Y);

            put(" Y value = "); put(Lower_Left_Y,4,10); new_line;
            put(" X value = "); put(Lower_Left_X,4,10); put(" (Y/N)?");

            get(Answer); new_line;

            case Answer is
            when 'y' | 'Y' => exit LOWER_LEFT;
            when others    => null;
            end case;

         end loop LOWER_LEFT;

      when 2 =>

         SIZE:
         loop

            new_line;
            put_line(" Enter size of box (1..255) => "); get(Box_Size);
            put(" Box size = "); put(Box_Size,3,10); put(" (Y/N)?");
            get(Answer);

            case Answer is
            when 'y' | 'Y' => exit SIZE;
            when others    => null;
            end case;

         end loop SIZE;

      when others =>

         exit BOX;

      end case;

   end loop BOX;

   put_line(" %REM - Setting values to zero...");

   for X in Input'first(1)..Input'last(1)
   loop
      for Y in Input'first(2)..Input'last(2)
      loop

         if X - 128 > Lower_Left_X
```

AD-A190 679 FEATURE EXTRACTION USING THE HOUGH TRANSFORM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING M L HILL DEC 87 AFIT/GE/ENG/87D-24 2/2

UNCLASSIFIED F/G 12/6 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

```
          and X - 128 < Lower_Left_X + Box_Size then

          if 256 - Y - 128 > Lower_Left_Y
             and 256 - Y - 128 < Lower_Left_Y + Box_Size then

             Input(X,Y) := 0;

          end if;

       end if;

    end loop;
 end loop;

when others => null;

end case;

new_line(2);
put_line(" ENTER NAME OF SSI EDGE FILE");
GET_FILENAME( Name, Char_Count );
SAVE_SSI_IMAGE( Name, Char_Count, Input );

new_line(2);
put_line(" *** END OCCLUDE ***");
new_line(2);

end OCCLUDE;
```

```
--*****************************************************************************
--
--                      ROTATE_EDGE IMAGE (SSI - Format)
--
--.............................................................................
--
--  Purpose:  Rotate an image by theta
--
--*****************************************************************************

with DATA_STANDARD;     use DATA_STANDARD;
with FILE_IO;           use FILE_IO;
with SSI_IO;            use SSI_IO;
with text_io;           use text_io;
with integer_text_io;   use integer_text_io;
with float_math_lib;    use float_math_lib;

procedure ROTATE_EDGE is

Input, Output          : DATA_STANDARD.Image_Array_2d(1..256,1..256);
Name                   : DATA_STANDARD.Line_Form;

Edge_Value             : constant integer := 255;
X_Center, Y_Center     : constant float   := 128.0;

Angle                  : integer range 0..359;
X, Y, Radius,
Theta_In, Theta_Out    : float;
X_Out, Y_Out           : integer;

Char_Count             : natural;
Answer                 : character;

begin -- ROTATE_EDGE

-- [1] Get image

new_line;
put_line(" NAME OF THE INPUT SSI FILE?");
GET_FILENAME(Name, Char_Count);
READ_SSI_IMAGE(Name, Char_Count, Input);

-- [2] Set output image to zero

for Y1 in Output'first(2)..Output'last(2)
loop
   for X1 in Output'first(1)..Output'last(1)
   loop
      Output(X1,Y1) := 0;
   end loop;
end loop;

-- [2] Get rotation angle
```

```
GET_ANGLE:
loop

   new_line;
   put(" ENTER ROTATION ANGLE (0 .. 359) > "); get(Angle); new_line;
   put(" Angle = "); put(Angle, 3, 10); put("? (Y/N) => ");
   get(Answer); new_line;

   case Answer is
   when 'Y' | 'y' =>
      exit GET_ANGLE;
   when others =>
         null;
   end case;

end loop GET_ANGLE;

-- [3] Shift and save in SSI format

put_line(" %REM - Rotating edge image...");

for Y1 in Input'first(2)..Input'last(2)
loop
   for X1 in Input'first(1)..Input'last(1)
   loop
      if Input(X1,Y1) = Edge_Value then

         X := float(X1) - X_Center;
         Y := float(Y1) - Y_Center;

         Radius := sqrt(X**2 + Y**2);
         Theta_In := atan2d(Y,X);

         Theta_Out := Theta_In + float(Angle);

         X_Out := integer(Radius * cosd(Theta_Out) + X_Center);
         Y_Out := integer(Radius * sind(Theta_Out) + Y_Center);

         if X_Out in Output'range(1) and Y_Out in Output'range(2) then

            Output(X_Out, Y_Out) := Edge_Value;

         end if;

      end if;
   end loop;
end loop;

new_line;
put_line(" NAME OF OUTPUT SSI FILE?");
GET_FILENAME(Name, Char_Count);
SAVE_SSI_IMAGE(Name, Char_Count, Output);
```

```
new_line;
put_line(" *** END ROTATE_EDGE ***");
new_line;

end ROTATE_EDGE;
```

```
--***************************************************************************
--
--                    MOVE_EDGE IMAGE (SSI - Format)
--
--...........................................................................
--
-- Purpose: Translate an edge image by input x and y increments
--
--***************************************************************************

with DATA_STANDARD;      use DATA_STANDARD;
with FILE_IO;            use FILE_IO;
with SSI_IO;             use SSI_IO;
with text_io;            use text_io;
with integer_text_io;    use integer_text_io;

procedure MOVE_EDGE is

Input, Output           : DATA_STANDARD.Image_Array_2d(1..256,1..256);
Name                    : DATA_STANDARD.Line_Form;
X_Shift, Y_Shift        : integer range -128..128;
Edge_Value              : constant integer := 255;
Char_Count              : natural;
Answer                  : character;

begin -- MOVE_EDGE

-- [1] Get image

new_line;
put_line(" NAME OF THE INPUT SSI FILE?");
GET_FILENAME(Name, Char_Count);
READ_SSI_IMAGE(Name, Char_Count, Input);

-- [2] Set output image to zero

for Y in Output'first(2)..Output'last(2)
loop
   for X in Output'first(1)..Output'last(1)
   loop
      Output(X,Y) := 0;
   end loop;
end loop;

-- [2] Get shift values

begin

GET_X_SHIFT:
loop
   new_line;
   put_line(" ENTER SHIFT IN X (-128 .. 128)");
   get(X_Shift);
```

```
   put(" X SHIFT = ");
   put(X_Shift);
   put("? (Y/N) => ");
   get(Answer);

   case Answer is
      when 'Y' | 'y' =>
         exit GET_X_SHIFT;
      when others =>
         null;
   end case;

end loop GET_X_SHIFT;

GET_Y_SHIFT:
loop
   new_line;
   put_line(" ENTER SHIFT IN Y (-128 .. 128)");
   get(Y_Shift);
   put(" Y SHIFT = ");
   put(Y_Shift);
   put("? (Y/N) => ");
   get(Answer);

   case Answer is
      when 'Y' | 'y' =>
         exit GET_Y_SHIFT;
      when others =>
         null;
   end case;

end loop GET_Y_SHIFT;

exception
   when Data_Error =>
      put_line(" Invalid entry.");
      put_line(" Enter integer value (-128 .. 128).");
end;

-- [3] Shift and save in SSI format

put_line(" %REM - Shifting edge points...");

for Y in Input'first(2)..Input'last(2)
loop
   for X in Input'first(1)..Input'last(1)
   loop
      if Input(X,Y) = Edge_Value then
         Output(X + X_Shift, Y + Y_Shift) := Edge_Value;
      end if;
   end loop;
end loop;
```

```
new_line;
put_line(" NAME OF OUTPUT SSI FILE?");
GET_FILENAME(Name, Char_Count);
SAVE_SSI_IMAGE(Name, Char_Count, Output);

new_line;
put_line(" *** END MOVE_EDGE ***");
new_line;

end MOVE_EDGE;
```

```
--***************************************************************************
--
--                              EXPAND
--
--...........................................................................
--
--  Purpose:  Perform linear expansion of image for display
--
--***************************************************************************

with DATA_STANDARD;      use DATA_STANDARD;
with FILE_IO;            use FILE_IO;
with SSI_IO;             use SSI_IO;
with text_io;            use text_io;
with integer_text_io;    use integer_text_io;
with float_math_lib;     use float_math_lib;

procedure EXPAND is

Input            : DATA_STANDARD.Image_Array_2d(0..359,0..181);
Max              : integer := 0;
Min              : integer := 300;
Name             : DATA_STANDARD.Line_Form;
Char_Count       : integer;

begin -- EXPAND

new_line;
put_line("NAME OF HS SSI INPUT FILE?");
GET_FILENAME(Name, Char_Count);
READ_SSI_IMAGE(Name, Char_Count, Input);

new_line;
put_line(" $REM - Perfoming linear expansion...");

for X1 in Input'first(1)..Input'last(1)
loop
   for Y1 in Input'first(2)..Input'last(2)
   loop
      if Max < Input(X1,Y1) then
         Max := Input(X1,Y1);
      end if;
      if Min > Input(X1,Y1) then
         Min := Input(X1,Y1);
      end if;
   end loop;
end loop;

for X1 in Input'first(1)..Input'last(1)
loop
   for Y1 in Input'first(2)..Input'last(2)
   loop
      Input(X1,Y1) := ( 255 * ( Input(X1,Y1) - Min ) ) / ( Max - Min );
```

```
   end loop;
end loop;

new_line;
put_line("NAME OF SSI OUTPUT FILE");
GET_FILENAME(Name, Char_Count);
SAVE_SSI_IMAGE(Name, Char_Count, Input);

new_line;
put_line(" *** END EXPAND ***");
new_line;

end EXPAND;
```

BIBLIOGRAPHY

1. Hough, P. V. C. *Method Means for Recognizing Complex Patterns.* United States Patent 3,069,654, 18 December 1962.

2. Duda, R. O. and P. E. Hart. "Use of the Hough transform to detect lines and curves in pictures," *Communcations of the Association of Computing Machines*, 5: 11-15 (1972).

3. Duda, R. O. and P. E. Hart. *Pattern Recognition and Scene Analysis.* New York: John Wiley & Sons, 1973.

4. Shapiro, S. D. "Transformations for the Computer Detection of Curves in Noisy Pictures," *Computer Graphics and Image Processing*, 4: 328-338 (1975).

5. Shapiro, S. D. "Feature Space Transforms For Curve Detection," *Pattern Recognition*, 10: 129-143 (1978).

6. Ballard, D. H. "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, 13: 111-122 (1981).

7. Ballard, D. H. and C. H. Brown. *Computer Vision.* Englewood Cliffs, New Jersey: Prentice-Hall, 1982.

8. Casasent, D. and R. Krishnapuram. "Curved Object Location by Hough Transformations and Inversions," *Pattern Recognition*, 20: 181-188 (1987).

9. Krishnapuram, R. and D. Casasent. "Hough Space Transformations for Discrimination and Distortion Estimation," *Computer Vision, Graphics and Image Processing*, 38: 299-316 (1987).

10. Boyce, J. F., G. A. Jones and V. F. Leavers. "An Implementation of the Hough Transform for Line and Circle Location," *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE)*, 808, Inverse Problems in Optics: 69-75 (1987).

11. Steier, W. H. and R. K. Shori. "Optical Hough Transform," *Applied Optics*, 25: 2734-2738 (15 August 1986).

12. Ambs, P., S. H. Lee, Q. Tian and Y. Fainman. "Optical Implementation of the Hough Transform by Matrix of Holograms," *Applied Optics*, 25: 4039-4045 (15 November 1986).

13. Eichmann, G. and B.Z. Dong. "Coherent Optical Production of the Hough Transform," *Applied Optics*, 22: 830-834 (15 March 1983).

14. Deans, S. R. *The Radon Transform and Some of Its Applications*. New York: John Wiley & Sons, 1983.

15. Casasent, D. et al. "Real-Time Deformation Invariant Optical Pattern Recognition Using Coordinate Transformations," *Applied Optics*, 26: 938-942 (1 March 1987).

16. Casasent, D. and D. Psaltis. "Deformation Invariant, Space-Variant Optical Pattern Recognition," *Progress in Optics XVI*. 290- 356, 1978.

17. Lee, A. J. and Casasent D. P. "Computer Generated Hologram Recording Using a Laser Printer," *Applied Optics*, 26: 136-138.

18. Bryngdahl, O. "Geometrical Transformations in Optics," *Journal of the Optical Society of America*, 64: 1092-1099 (August 1974).

19. Jenson, A. S., L. Lindvold and E. Rasmussen. "Transformation of Image Positions, Rotations, and Sizes in Shift Parameters," *Applied Optics*, 26: 1775-1781 (1 May 1987).

20. Tong, C. *Target Segmentation and Image Enhancement Through Multisensor Data Fusion*. MS Thesis AFIT/GE/ENG/86D-55. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD-A178875).

21. Tong, C *et al*. "Multisensor Data Fusion of Laser Radar and Forward Looking Infrared (FLIR) for Target Segmentation and Enhancement," *Proceedings of the SPIE*, 782. 10-19 (1987).

22. Kobel, M and T. Martin. *Distortion Invariant Pattern Recognition in Non-Random Noise*. MS Thesis AFIT/GE/ENG/86D-20. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD-A177598).

23. Ruck D. *Multisensor Target Detection and Classification*. MS Thesis AFIT/GE/ENG/86D-56. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.

Vita

Captain Marvin L. Hill was born on 19 July 1960 in Savanna, Illinois. He graduated valedictorian from high school in Thomson, Illinois, in 1978 and attended the University of Illinois, from which he received the degree of Bachelor of Science in Electrical Engineering in May 1982. Upon graduation, he received a commission in the USAF through the ROTC program. He was called to active duty in October 1982. Upon completion of communications-electronics engineer technical training at Keesler AFB, Mississippi, he the served as evaluation team chief in the Air Traffic Control Communications Evaluation Section and section chief of the Weather Radar Evaluation Section at the 1866th Facility Checking Squadron, Scott AFB, Illinois until entering the School of Engineering, Air Force Institute of Technology, in May 1986.

Permanent address: Route 1, Box 43
Thomson, Illinois 61285

NCLASSIFIED
URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

REPORT SECURITY CLASSIFICATION NCLASSIFIED	1b. RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
DECLASSIFICATION/DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S) FIT/GE/ENG/87D-24	5. MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION chool of Engineering — 6b. OFFICE SYMBOL *(If applicable)* AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
ADDRESS *(City, State, and ZIP Code)* ir Force Institute of Technology right-Patterson AFB OH 45433-6583	7b. ADDRESS *(City, State, and ZIP Code)*
NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

TITLE *(Include Security Classification)*
FEATURE EXTRACTION USING THE HOUGH TRANSFORM

12. PERSONAL AUTHOR(S)
Marvin L. Hill, B.S., Capt, USAF

3a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
MS Thesis	FROM ______ TO ______	1987 December	107

6. SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	Hough Transform, Pattern Recognition Computer-Generated Holograms
12	09		

ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Thesis Chairman: Steven K. Rogers, Captain, USAF
Associate Professor of Electrical Engineering

Approved for public release: IAW AFR 190-1[illegible]
LYNN E. WOLAVER 14 Mar 88
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH [illegible]

DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
NAME OF RESPONSIBLE INDIVIDUAL teven K. Rogers, Captain, USAF	22b. TELEPHONE *(Include Area Code)* (513) 255-6027 — 22c. OFFICE SYMBOL AFIT/ENG

Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

19. This thesis applied the normal straight line parameterization of the Hough transform to a variety of images using the accumulator method. Simple inputs were used initially to illustrate the distortion characteristics of the Hough transform due to rotations, scales and translations of an input. Making use of work performed by D. Casasent and R. Krishnapuram of Carnegie-Mellon University, the Hough transform was then applied to segmented and edged doppler images. A distort-and-compare routine, which makes use of the Hough space distortion characteristics, was implemented in the Hough transform domain to estimate input space characteristics of an object.

Next, the Hough transform, generated using Fourier transform techniques, was applied to some of the same inputs to demonstrate that the accumulator method is actually a discrete version of the continuous Hough transform. An unsuccessful attempt at implementing the continuous Hough transform using computer-generated interferograms was outlined. A method of implementing the continous Hough transform and its inverse using phase filters was presented as a suggestion for further research.

END
DATE
FILMED
4-88
DTIC